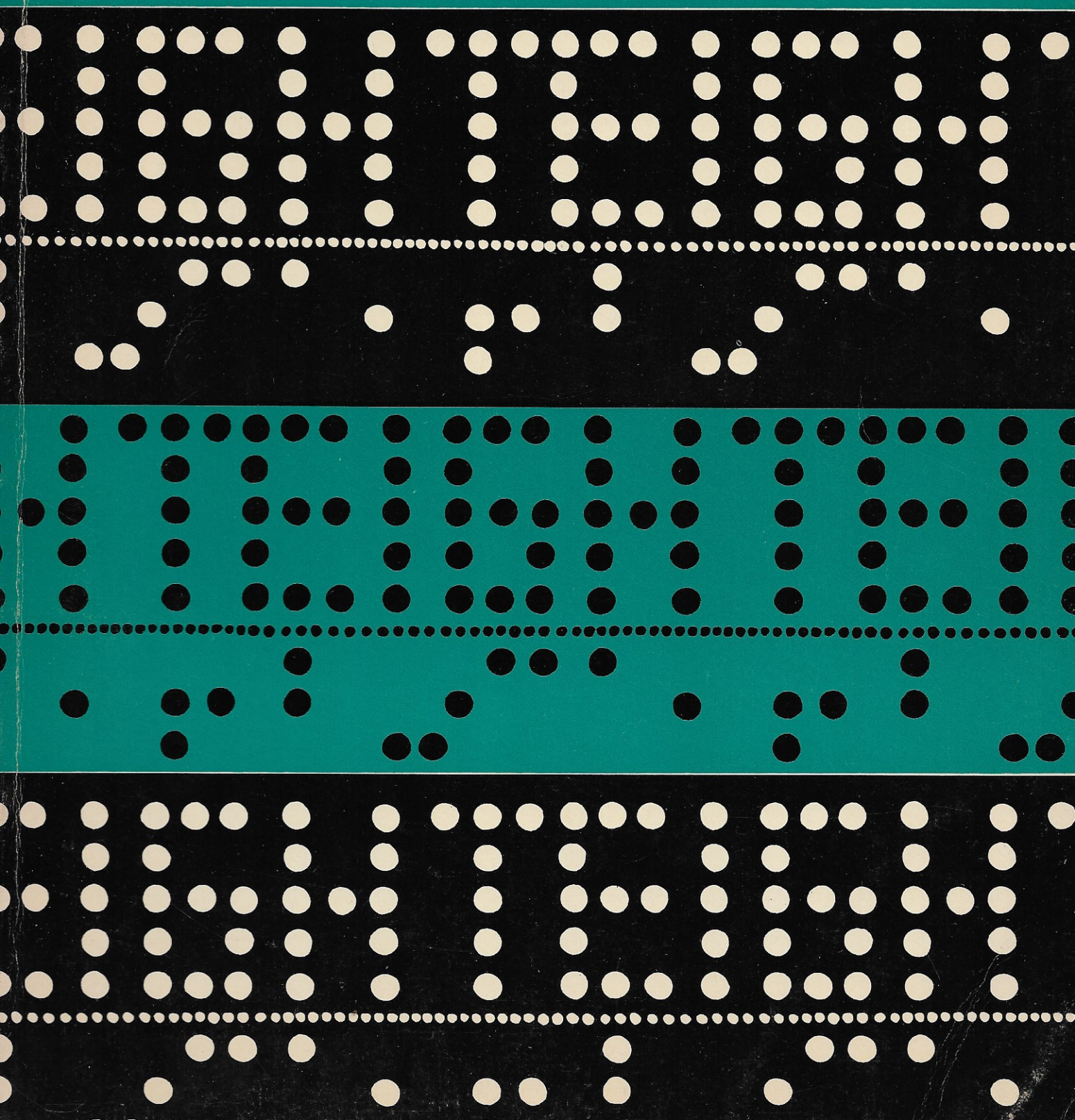
M100 8

THE OPEN UNIVERSITY

Curso Básico de Matemáticas Unidad 8

Computación I



The Open University

Curso Básico de Matemáticas Unidad 8

COMPUTACION I

Preparado por el Equipo del Curso Básico de Matemáticas

Traductor:
Nicolás Civetta, Profesor de Matemáticas
Universidad de Mérida, Venezuela

libros mcgraw-hill

méxico panamá madrid bogotá sao paulo nueva york
londres toronto sidney johannesburg
dusseldorf singapur

Queda terminantemente prohibido reproducir este libro, total
o parcialmente, sin autorización escrita del editor

RESERVADOS TODOS LOS DERECHOS (D. R.)

Copyright © 1974 respecto de la edición en idioma español, por
EDITORIAL McGRAW-HILL LATINOAMERICANA, S. A.
Colón, República de Panamá

0-07-090767-6

Traducido de COMPUTING I
Copyright © 1971 by the Open University Press
Bletchley, Bucks, England

IMPRESO EN COLOMBIA PRINTED IN COLOMBIA

El concepto de enseñanza abierta desarrollado por la Open University utiliza materiales en forma de libros, películas y cintas magnetofónicas que el estudiante utiliza *fuera* de la institución. *Sin embargo, estos materiales también pueden utilizarse dentro de programas tradicionales.* Los componentes de los programas desarrollados por la Open University son:

LIBROS: Todos los libros tienen el mismo formato, y fueron escritos, diseñados y producidos por especialistas de la Open University. La mayor parte de los libros están profusamente ilustrados con diagramas y fotografías —muchas a colores— y tienen un diseño fácil y atractivo para el estudiante. Cada libro contiene objetivos de aprendizaje muy explícitos, y cuestionarios de autoevaluación, así como respuestas y comentarios a la autoevaluación, permitiendo así al estudiante, darse cuenta de su propio progreso. *Todos estos libros pueden utilizarse como textos —ya sea en forma individual, o uniendo varias unidades— o como material de referencia.*

PELICULAS: Las películas han sido preparadas como *material suplementario y de refuerzo* para permitir al profesor mayor libertad de trabajo con cada estudiante. Las películas están en 16 mm, y en su mayor parte son en blanco y negro. A la fecha de publicación de estos libros sólo están a la venta en idioma inglés.

CINTAS MAGNETOFONICAS: Al igual que las películas, estas cintas fueron preparadas como *material suplementario* y para ser utilizadas por el estudiante en su propia casa. A la fecha de publicación de estos libros sólo están a la venta en idioma inglés.

NOTA A LOS MAESTROS: Se han mantenido todas las referencias a otros elementos del programa en los materiales impresos, por considerarse que no obstaculizan la utilización de los libros como textos independientes. Los profesores que no tengan acceso a los materiales audio-visuales fácilmente podrán sustituirlos con sus propios experimentos o explicaciones.

Esta obra se imprimió en los Talleres Gráficos de Carvajal & Cía.
de Cali, Colombia, en el mes de abril de 1974. Se imprimieron 5000
ejemplares en papel offset de 90 g/m².

Tabla de materias

Página

	Una observación para el estudio de este texto	v
	Objetivos	v
	Diagrama estructural	vi
	Glosario	vii
	Bibliografía	x
8.1	**Introducción**	1
8.1.1	Computadores y computación	1
8.1.2	Resolución de problemas y matemáticas constructivas	2
8.1.3	El enfoque	3
8.2	**El concepto de algoritmo**	3
8.2.1	Método de solución	3
8.2.2	Definición formal de algoritmo	6
8.2.3	Descripción de un algoritmo por medio de un diagrama de flujos	11
8.3	**Concepto de máquina lógica**	23
8.3.0	Introducción	23
8.3.1	El concepto de variable: Elemento de almacenamiento	23
8.3.2	Introducción al laboratorio de programación	24
8.3.3	Laboratorio de programación: Parte 1, constantes, variables, y asignaciones	26
	Ejercicio práctico 1	28
8.3.4	El concepto de ejecución de un trabajo: Elemento de procesamiento	29
8.3.5	Laboratorio de programación: Parte 2, operaciones aritméticas sencillas	32
	Ejercicio práctico 2	34
8.3.6	Evaluación de expresiones	35
8.3.7	Laboratorio de programación: Parte 3, expresiones aritméticas en lenguaje BASIC	39
	Ejercicio práctico 3	40
8.3.8	Comunicación con el usuario	41
8.3.9	Laboratorio de programación: Parte 4, entrada y salida de valores numéricos	42
	Ejercicio práctico 4	45
8.3.10	Concepto de programa almacenado	46
8.3.11	Laboratorio de programación: Parte 5, ensamblaje de un programa en lenguaje BASIC	48
	Ejercicio práctico 5	53

OBSERVACIONES AL ALUMNO

Al comienzo de cada texto encontrará lo siguiente.

1. UNA PRESENTACION DE NUESTROS OBJETIVOS en la unidad, la cual puede usar para comprobar si ha completado el curso satisfactoriamente.
2. UN DIAGRAMA ESTRUCTURAL que le muestra cómo están relacionadas las diferentes secciones de la unidad. El diagrama consta de:
   a) *cuadros rojos* que muestran la secuencia principal.
   b) *cuadros negros punteados* (a la izquierda del rojo) que le muestran lo que esperamos ya sepa antes de fijarse un estudio detallado.
   c) *cuadros negros* (a la derecha del rojo) que le ayudan a escoger temas que puede posponer u omitir si tiene poco tiempo.
3. UN GLOSARIO de las definiciones, notaciones, etc., usadas en la unidad como referencia inmediata.
4. UNA BIBLIOGRAFIA de los libros que puede consultar cuando haya acabado el trabajo de la unidad.
5. CONCLUSIONES Y RESUMEN del trabajo realizado al final de cada texto.

NUMERACION E INDICADORES

Dentro de cualquier sección nos referimos a los temas por medio de un número. Al referirnos al mismo tema en otra sección, citamos cuatro números: número de la unidad, número del capítulo o parte, número de sección y número del tema. Por ejemplo, "Ecuación 3" si aparece en la misma sección, y "Ecuación 4.3.1.3", si está en otra sección. Los tres primeros números se citan en la parte superior de la página. Creemos que este es un sistema claro y fácil para consultar cualquier tema. Los indicadores serán una guía valiosa y le ahorrarán tiempo.

En la margen derecha del texto hemos tratado de indicar la clase del material. Los términos usados son tan claros que no necesitan explicación. Por ejemplo, "Discusión" significa que una vez que haya entendido el tema, no necesita de nuevo recurrir a esta parte del texto. Estos INDICADORES tienen por objeto ayudarlo a localizar rápidamente las partes importantes cuando esté repasando o tenga que omitir parte del texto por falta de tiempo.

También hemos tratado de indicar la importancia de los temas por el sistema de estrellas.

*** Esto es muy importante. Usted debe entenderlo claramente. En algunos casos lo usará con tanta frecuencia, que se familiarizará con ello sin necesidad de memorizarlo deliberadamente. El punto principal de este curso es no aprenderse los datos de memoria; nosotros creemos que es más importante que entienda el trabajo y sea capaz de aplicarlo en la solución de los problemas.

** Este es de menor importancia, pero tiene material al cual nos referiremos más adelante.

* Esto generalmente no lo guía a trabajos adicionales en este curso y lo puede omitir si tiene poco tiempo.

■ Indica el final de cada ejemplo, ejercicio o solución.

Una observación para el estudio de este texto

Se deben estudiar las dos primeras secciones de esta unidad por el procedimiento normal. Una "lectura completa" de la Sección 8.3 culminará su trabajo en este texto.

Un estudio minucioso de la tercera sección, incluyendo el desarrollo de los detalles del trabajo práctico, se hará de acuerdo con sus capacidades durante cinco semanas aproximadamente. Como orientación se sugiere que emplee hasta una hora (no más de una hora) en cada una de las cinco partes del texto de Laboratorio de programación y que gaste hasta 15 minutos (no más de 15 minutos) de este tiempo, desarrollando el trabajo práctico en una estación terminal o copiando y verificando su solución en una hoja de codificación.

Objetivos

Después de estudiar las Secciones 8.1 y 8.2 de este texto, obtendrá:

(i) una comprensión sobre el enfoque sistemático, etapa por etapa, para la resolución de problemas, que es el ingrediente esencial en el proceso de utilización del computador como herramienta para la resolución de problemas;

(ii) un concepto intuitivo sobre la naturaleza de la computación y de los problemas involucrados en el uso del computador como herramienta de trabajo en la resolución de problemas;

(iii) una comprensión adecuada de los conceptos básicos de la computación, la cual formará una base para su trabajo práctico de computación.

Después de un estudio preliminar de la Sección 8.3 de este texto tendrá:

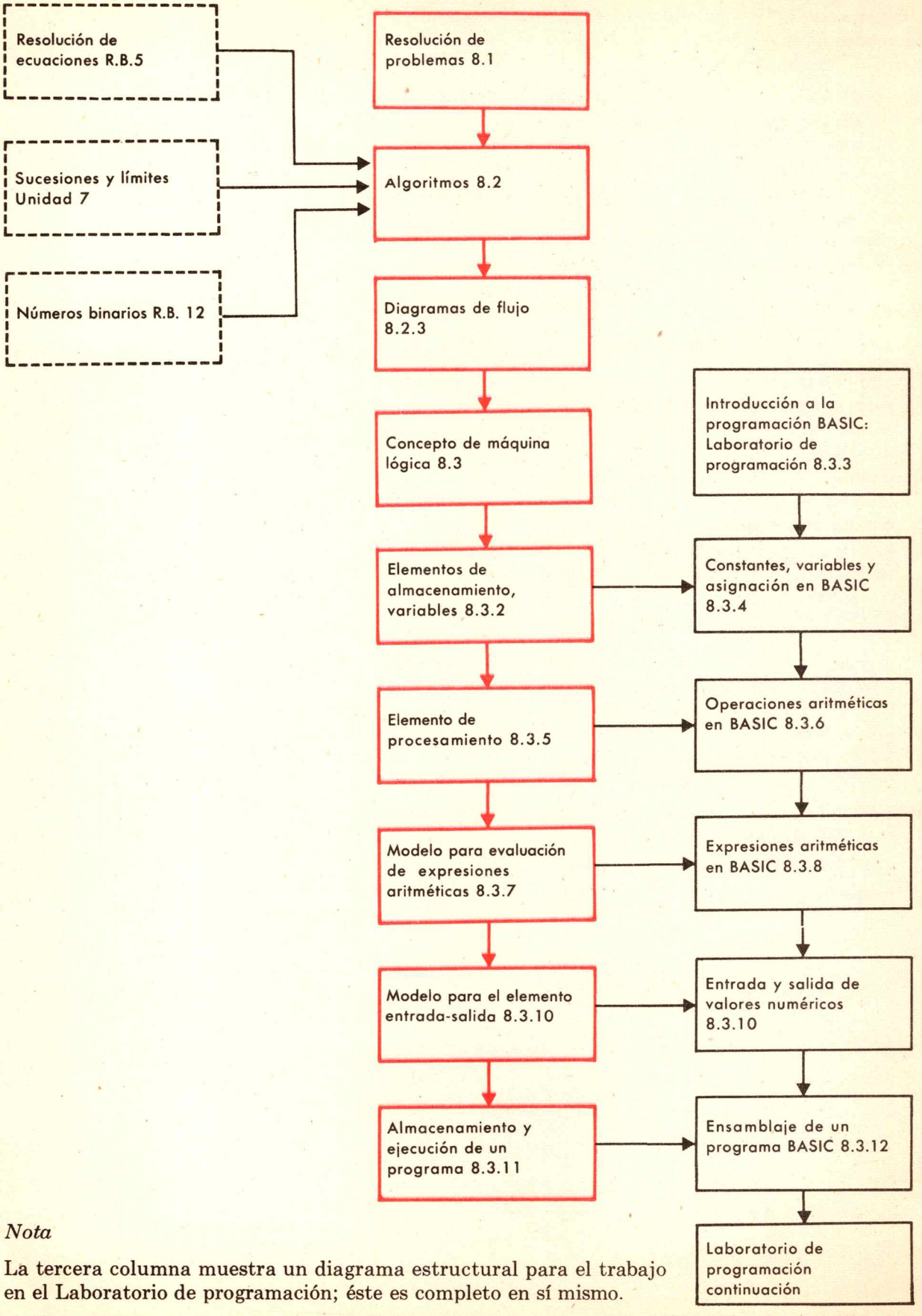
(iv) una comprensión intuitiva de la organización funcional de un computador digital basada en el entendimiento de un modelo conceptual aplicado a un sistema de computadores;

(v) una comprensión de los elementos del lenguaje de programación BASIC interpretados en función del modelo conceptual aplicado a un sistema de computadores.

Después de completar el trabajo práctico asociado a la tercera sección de este texto obtendrá:

(vi) la capacidad para escribir programas sencillos en el lenguaje de programación BASIC.

Diagrama estructural



Nota

La tercera columna muestra un diagrama estructural para el trabajo en el Laboratorio de programación; éste es completo en sí mismo.

Glosario

		Página
ALGORITMO	Un *algoritmo* es un *procedimiento* por el cual, después de un número *finito* de etapas, se obtiene una solución del problema, o se encuentra que no existe solución y la ejecución se detiene.	7
APARATO	Un *aparato* es una máquina capaz de interpretar una sucesión de *instrucciones*, expresadas en algún *lenguaje* el cual especifica la ejecución de ciertas operaciones por el aparato.	4
BASIC	El *basic* es un *lenguaje de programación* sencillo utilizado en el Curso Básico (Beginner's All-purpose Symbolic Instruction Code).	25
CIRCULO CONECTIVO	Cuando debe haber una interrupción en un *diagrama de flujos*, se usa el *círculo conectivo* con una marca. Dos o más de tales símbolos representan el mismo punto en el diagrama de flujos.	14
DECISION BINARIA	Una *decisión binaria* es la decisión que se toma entre *dos* alternativas.	13
DECLARAR UNA VARIABLE	*Declarar una variable* consiste en poner un nombre a un *sitio de almacenamiento* que no ha sido previamente marcado.	24
DIAGRAMA DE FLUJOS	Un *diagrama de flujos* es una representación pictórica de un *algoritmo* y en la cual los pasos que se deben ejecutar están indicados por recuadros con *instrucciones* escritas dentro, y el orden de las *operaciones* está especificado por las flechas que los conectan.	11
DIRECCION	El nombre, X, del sitio que contiene el valor actual de la *variable x* se llama *dirección* del sitio.	26
ELEMENTO DE ALMACENAMIENTO	El *elemento de almacenamiento* en un *modelo* es una colección de sitios para los cuales el modelo puede recibir y ejecutar las siguientes *instrucciones:* (i) *Declarar una variable.* (ii) Dar un valor a una *variable*, remplazando su valor anterior. (iii) Copiar el valor de una variable. (iv) Liberar un *sitio de almacenamiento*; esto es, remover la marca y borrar el valor que le está asociado.	24
ELEMENTO DE PROCESAMIENTO	El *elemento de procesamiento* de un *modelo* es un conjunto de servicios que nos capacitan para llevar a cabo instrucciones individuales y para seguir una ruta a través de un *algoritmo.*	29
EXPONENTE	Ver *formato normalizado de punto flotante.*	

		Página
FORMATO NORMALIZADO DE PUNTO FLOTANTE	El *formato normalizado de punto flotante* es la representación de un número decimal, x, en la forma mEe, donde $x = m \times 10^e$ y $m \in [1, 10[$. m es llamada la *mantisa*, 10 es la base y e es el *exponente*.	40
INSTRUCCIONES	Las *instrucciones* son directivas dadas a un *aparato* en un *lenguaje* que éste pueda interpretar, las cuales especifican la ejecución de ciertas operaciones por el *aparato*.	3
LENGUAJE	Un *lenguaje* es un conjunto de símbolos utilizados para dar *instrucciones* a un *aparato*.	4
LENGUAJE DE PROGRAMACION	Un *lenguaje de programación* es un *lenguaje* en el cual son escritas las *instrucciones* de un *programa*.	25
MANTISA	Ver *formato normalizado de punto flotante*.	
MAQUINA LOGICA	Una *máquina lógica* es una máquina cuyo diseño está basado en las leyes de la lógica por ejemplo, un computador automático.	1
MODELO	Un *modelo* es la representación simplificada de algo, esto es, una ayuda visual. Por ejemplo, un conjunto de agujeros que contengan tiras de papel es un modelo que nos sirve para visualizar un conjunto de *sitios de almacenamiento* que contienen datos.	1, 16, 23, 29
METODO CONSTRUCTIVO	Un *método constructivo* es un *procedimiento* que nos capacita para construir la solución de un problema por medio de la aplicación sistemática de un número finito de etapas.	2
NUMERO DE LINEA	Un *número de línea* es un número entero no especificado, comprendido entre 1 y 9999 el cual debe aparecer al comienzo de cada *instrucción* en el lenguaje *basic*.	49
OPERACIONES	Las *operaciones* son los procesos que pueden ser ejecutados por un *aparato*.	4
ORDEN	Una *orden* es una directiva dada a un computador, la cual se ejecuta de inmediato y no forma parte del *programa* almacenado.	48
PROCEDIMIENTO	Un *procedimiento* para la resolución de un problema, P, en un *aparato*, D, es una especificación (hecha en un *lenguaje* aceptable por D) de una sucesión finita de pasos (ejecutable por D) tal que, si D ejecuta las etapas en el orden prescrito, entonces se obtendrá una solución de P, si esta solución existe.	4
PROGRAMA	Un *programa* es una sucesión de *instrucciones* que deben ser ejecutadas por un computador.	25
PROGRAMACION	La tarea de planear y escribir una serie de *instrucciones* con el fin de llevar a cabo una determinada tarea (esto es, que corresponda	25

		Página
	a un *algoritmo* particular) se llama *programación.*	
RECUADRO DE ENTRADA	En un *diagrama de flujos,* la información destinada a servir como datos de entrada al *algoritmo* está especificada en un *recuadro de entrada.*	14
RECUADRO DE SALIDA	En un *diagrama de flujos,* la información que da el resultado suministrado por el *algoritmo* está especificada en un *recuadro de salida.*	14
SITIO DE ALMACENAMIENTO	Un *sitio de almacenamiento* en un *modelo* conceptual es una localización en la cual almacenamos el valor actual de una variable.	24
SUBRUTINA	Una *subrutina* es una sección de un *programa* la cual es discutida o *diagramada* en detalle en alguna otra parte.	20
VARIABLE	Una parte de una información, número, sucesión, etc., que está siendo manipulada por un *algoritmo,* se llama una *variable.*	23

Bibliografía

S. H. Hollingdale y G. C. Tootill, *Electronic Computers* (Penguin Books, 1965).
Los Capítulos 1-3 ofrecen una útil introducción a los computadores y a la computación, basada en una revisión histórica de la computación.

G. Polya, *How to Solve it,* Open University Ed. (Doubleday Anchor Books, 1970).
La parte I analiza la resolución de problemas en relación con los problemas de matemáticas. Este material hace parte de la base de nuestras exposiciones sobre procedimientos y algoritmos. Este libro es fundamental para este curso y usted debiera adquirir un ejemplar; en este libro nos referimos a él como *Polya.*

R. R. Korfhage, *Logic and Algoritms: with applications to the Computer and Information Sciences* (John Wiley, 1966).
El Capítulo 5, Algoritmos y Máquinas de Computación, define procedimientos y algoritmos y llega a relacionarlos con las máquinas abstractas (autómatas finitos) y los computadores digitales. Este material es complementario a la exposición incluida en el presente texto.

8.1 INTRODUCCION

8.1

8.1.1 Computadores y computación

8.1.1

Introducción
* *

Han pasado apenas algo más de dos décadas desde el advenimiento de los primeros computadores de alta velocidad y totalmente automáticos; sin embargo, ya se están utilizando ampliamente y están destinados indudablemente a ejercer una considerable influencia sobre el futuro desarrollo de muchos de los aspectos de la sociedad en que vivimos.

El deseo de hacer la computación menos laboriosa, ha motivado el desarrollo y la construcción de numerosas calculadoras mecánicas como ayuda al trabajo de computación. El computador automático moderno representa la última adición a una larga línea de aparatos calculadores, que se remonta por miles de años hasta el quipu y el ábaco.

En este corto curso no tenemos tiempo para exponer la historia y evolución de los aparatos de calcular que condujeron al desarrollo de los modernos implementos y culminaron con el computador moderno. Sin embargo, se encuentran en la literatura un cierto número de exposiciones adaptadas para el nivel de este curso. El primer libro de la bibliografía (Capítulos 1-3) está especialmente indicado como lectura básica si usted está interesado en una guía histórica del desarrollo de la computación y los computadores.

Esta unidad y la *Unidad 20, Computación II* podrían ser tituladas colectivamente "La ciencia de la computación - Una Apreciación", ya que nuestro objetivo principal es llegar en alguna forma a dar respuestas a las siguientes preguntas:

(i) ¿Qué es la ciencia de la computación?
(ii) ¿Por qué es importante la ciencia de la computación?

Antes de que podamos intentar formular cualquier clase de respuesta a alguna de estas preguntas, debemos llevar a cabo un estudio introductorio pero completo de la computación (como contraste a los computadores); esto es, debemos efectuar un estudio de las técnicas involucradas en la resolución de problemas con la ayuda del computador.

Es imposible describir sistemáticamente aquellas tareas que puedan ser ejecutadas por un computador y aquellas que no lo pueden ser. Es imposible por las siguientes razones: primero, un computador puede ejecutar únicamente una tarea que le haya sido completa y explícitamente especificada por quien lo usa (así sucede que algunas de las restricciones en los trabajos que un computador puede realizar están directamente relacionadas con la destreza del usuario para especificar el trabajo); segundo, los computadores (máquinas lógicas) son máquinas muy complejas y actualmente carecemos de las herramientas necesarias para analizar sus actividades. En esta unidad consideraremos ambos aspectos: el primero, por el estudio de los métodos que especifiquen formalmente la resolución de un problema; el último, por medio del estudio de un *modelo* conceptual sencillo de computador. Los conceptos abstractos introducidos en estas exposiciones obtendrán una realidad concreta por medio del trabajo práctico que deberá llevar a cabo en unión de ésta y de la segunda unidad de computación. Una vez completado el estudio de este texto y del trabajo práctico que lo acompaña, usted tendrá un conocimiento razonable e intuitivo de la naturaleza de la computación, de los problemas inherentes al uso del computador como herramienta para resolver problemas y de algunos de los usos potenciales y limitaciones de los computadores.

8.1.2 Resolución de problemas y matemáticas constructivas

8.1.2

Introducción
* *

Uno de los útiles subproductos asociados con cualquier estudio serio de la computación es que ofrece la oportunidad casi sin rival para investigar otras dos áreas de importancia fundamental en matemáticas, a saber, resolución de problemas y matemáticas constructivas. De hecho, el conocimiento que usted adquiere en estas áreas cuando estudia computación puede ser tan importante como el conocimiento adquirido acerca de la computación misma.

En esta unidad así como en la segunda unidad de computación, estaremos interesados en examinar las formas en las cuales la gente, con ayuda de aparatos a su disposición, resuelve problemas. El advenimiento de los computadores automáticos (máquinas lógicas) que pueden manipular formas muy generales de información de acuerdo con un conjunto preespecificado de instrucciones, motiva el estudio de varios aspectos de la *resolución automática de problemas.* Esto, a su vez, da motivos para considerar en detalle áreas completas de la resolución de problemas y, en particular, de los procesos de resolución de problemas utilizados en matemáticas. Se le recomienda leer por lo menos la parte I de *Polya** antes o después de estudiar esta unidad.

Definición 1
*

En matemáticas entendemos por **método constructivo**, a un procedimiento que nos capacita para construir la resolución de un problema por medio de la aplicación sistemática de un número *finito* de pasos. Hasta la época de Euler (primeras décadas del siglo XVIII), la mayor parte de las matemáticas estaba inspirada no sólo en interrogantes concretos y problemas, sino que también aspiraba a resolver dichos problemas directamente por medio de técnicas constructivas. A medida que los problemas crecían en amplitud y generalidad, las técnicas constructivas cedieron el paso a las técnicas no constructivas basadas únicamente en la lógica. Desde los primeros tiempos hasta la llegada del computador digital moderno, la velocidad con la cual el hombre ejecutaba sus cálculos se multiplicó por un factor igual quizá a 10 ó 100. A partir de la aparición del computador hasta nuestros días (aproximadamente dos décadas) la velocidad con la cual ejecutamos los cálculos se ha multiplicado por un millón. También estamos en capacidad de usar el computador para ejecutar un enorme número de manipulaciones individuales pre-especificadas sin intervención humana y manejar tipos de información mucho más complejos que los que pueden ser representados con el simple concepto de número. Como consecuencia de esto, el interés sobre las técnicas constructivas de todos los tipos ha aumentado enormemente. Esta tendencia no desplazará (por lo menos en matemáticas) la necesidad de técnicas lógico deductivas; pues una vez que hayamos formulado un método constructivo, deseamos determinar las condiciones exactas bajo las cuales el método da la solución del problema considerado. En computación, nos vemos reducidos a menudo a juzgar el comportamiento de un método desde un punto de vista pragmático; cuando el método ha sido verificado en un cierto número de casos especiales y ha resultado satisfactorio, se supone satisfactorio para todos los casos semejantes. Matemáticamente tal procedimiento es poco serio: debemos basar nuestro juicio en la prueba lógica con preferencia a la prueba empírica. Entonces debemos tratar de probar teoremas concernientes a la rectitud y otras propiedades relievantes de nuestros méto-

* Ver Bibliografía.

dos constructivos. La aplicación de técnicas lógicas con este fin está en su infancia en cuanto a métodos constructivos se refiere, y los problemas que se tienen que vencer son enormes. De todos modos, es importante que usted llegue a valorar los defectos y debilidades de las técnicas de verificación empírica de un método constructivo.

8.1.3 El enfoque

8.1.3

Introducción
* *

En esta unidad, comenzamos por considerar los problemas inherentes a la descripción de una técnica constructiva como base para un método automático de resolución de problemas. Llegamos a examinar una técnica formal para la descripción de un *algoritmo* (nombre dado a una forma especial de método constructivo) y, a continuación, estudiamos un modelo conceptual para una máquina lógica, el cual nos dará una base para describir la ejecución de un algoritmo. En esta sección final, presentamos las disponibilidades obtenibles a través del Servicio de Computación para el estudiante y sentamos los fundamentos para enfrentarnos a la computación práctica asociada al curso.

Para obtener un beneficio completo del estudio de este texto, haremos énfasis sobre nuestra recomendación general para todos los textos: haga un intento serio de resolver cada ejercicio (y, si tiene tiempo, intente resolver cada ejemplo del texto) antes de leer la solución que se le ha dado. La experiencia adquirida en resolver problemas particulares lo ayudará a entender los conceptos descritos en el texto.

8.2 EL CONCEPTO DE ALGORITMO

8.2

8.2.1 Método de solución

8.2.1

Discusión
* *

En esta sección estamos interesados en un aspecto particular de la segunda fase de la actividad de la resolución de problemas destacada por Polya en *How to Solve It*, a saber, la actividad de diseñar *un plan*. ¿Qué comprende esta actividad? Decimos con Polya (página 8):

"Tenemos un plan cuando sabemos, o sabemos por lo menos en líneas generales, qué cálculos, cómputos o construcciones debemos ejecutar con el fin de encontrar la incógnita".

En esta unidad no nos preocupamos por la totalidad de las técnicas por medio de las cuales puede obtenerse un método de solución, sino con las técnicas para diseñar un plan en tal forma que éste pueda ser especificado formalmente en función de un conjunto de *instrucciones* que pueda interpretar una máquina lógica. Si este objetivo se logra, entonces la fase siguiente de la actividad de resolver problemas, la fase que Polya llama *ejecución del plan*, se puede llevar a cabo sistemáticamente por una máquina lógica que *ejecuta* el plan especificado. Cuando tenemos una descripción formal de un método de solución, decimos que tenemos un *procedimiento* para la resolución del problema. En consecuencia, un *procedimiento* es una especificación formal de un plan para la resolución de un problema, en el sentido en el cual Polya utiliza el término.

Consideremos el problema de definir más explícitamente qué queremos decir con el concepto de un *procedimiento*. Debemos preguntarnos primero si existe o no existe una solución. Si se nos da un problema y somos capaces de concebir un plan para obtener su solución, entonces podemos estar en capacidad de decir inmediatamente si esta solución existe o no. Por ejemplo, si la existencia de una solución depende de la existencia de raíces *reales* de la ecuación cuadrática.

$$ax^2 + bx + c = 0, \qquad a, b, c \in R, \qquad a \neq 0$$ (Véase RB5)

entonces podemos discriminar entre aquellos casos para los cuales existe una solución y aquellos para los cuales no existe, encontrando el signo del número $(b^2 - 4ac)$ para los valores dados a, b, y c.

(Cuando $b^2 - 4ac$ es positivo o cero, la solución está dada por la fórmula $x = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$.) Sin embargo, las cosas no son siempre tan nítidas como ésta; a menudo puede ser necesario ejecutar parte del plan antes de ser capaces de decidir si existe o no, una solución. Algunas veces es imposible decidir, basados en el plan concebido, cuando existe o no una solución. Por ejemplo, si la existencia de una solución depende de la existencia del límite de una sucesión (véase la *Unidad 7, Sucesiones y límites I*), entonces no existe un procedimiento general que deba determinar si existe o no una solución.

Lo anterior nos lleva a definir la primera característica de un procedimiento:

(i) Si un problema tiene solución, entonces la ejecución de un procedimiento debe encontrar una solución; si, sin embargo, el problema no tiene solución, entonces un procedimiento puede o bien reconocer este hecho, o bien continuar indefinidamente. Un procedimiento no termina nunca con una solución no válida.

La segunda característica de un procedimiento se puede definir como sigue:

(ii) Un procedimiento debe ser compatible con el *aparato* utilizado para llevarlo a cabo. Esto quiere decir que el procedimiento se debe expresar en un *lenguaje* que pueda ser interpretado por el aparato y en función de las *operaciones* que el aparato pueda llevar a cabo (ejecutar).

Por ejemplo, supongamos que el problema consiste en la resolución de una ecuación; entonces, si el aparato es un hombre equipado con lápiz y papel, el procedimiento podría consistir en una sencilla resolución gráfica de la ecuación (tal como la descrita en la Sección 2.4.1 de la *Unidad 2, Errores y Exactitud*). Sin embargo, si el aparato consiste de un hombre y una calculadora de escritorio, puede formularse un procedimiento más sofisticado (por ejemplo, la técnica iterativa descrita también en la Sección 2.4.1 de la *Unidad 2*). En esos casos, el lenguaje utilizado para describir los procedimientos en la *Unidad 2* es bastante sofisticado, ya que, el principal elemento de procesamiento es un hombre. Siempre es importante considerar cómo aparear lo mejor posible al procedimiento con las capacidades del aparato que lo ejecuta: esto se aplica cuando se utiliza cualquier máquina lógica.

La tercera característica de un procedimiento se puede definir como sigue:

(iii) Un procedimiento consiste en un número finito de etapas que se deben ejecutar en un orden especificado.

Esta característica se basa en la suposición de que el aparato de procesamiento opera en etapas finitas cada una de las cuales tiene un comienzo y un final definidos.

Las tres características nos llevan a la definición formal de un *procedimiento*; esta definición se basa en la dada en el libro de R. R. Korfhage, *Logic and Algorithms* (véase la bibliografía).

Definición 1

Un **procedimiento** para la resolución de un problema, P, en un aparato, D, es una descripción en un lenguaje aceptable por D, de una sucesión de etapas finitas ejecutables por D; el ordenamiento de dichas etapas es tal que (dados los datos apropiados), si D ejecuta los pasos en el orden debido, se obtendrá una solución de P, si ésta existe. Si no existe solución para P, entonces la ejecución del procedimiento puede o no detectar este hecho.

La implementación de esta definición, con el fin de obtener un procedimiento definido formalmente a partir de un método de solución*, necesita un conocimiento y una comprensión completas tanto del *lenguaje* asociado al aparato como de las *operaciones* que éste es capaz de ejecutar. (El conocimiento, sin embargo, no necesita ser completo: uno puede utilizar una calculadora de escritorio para sumar sin saber qué más puede hacerse con ella.) Esto puede llegar a ser más claro para usted después de que haya visto un ejemplo. Consideremos un problema del cual usted ya conoce la respuesta: cómo encontrar el significado de una palabra cuyo significado le es poco familiar.

Ejemplo 1

Ejemplo 1

Por supuesto, usted sabe cómo encontrar una palabra en un diccionario —pero, ¿puede describir *exactamente* cómo hacerlo? ¿Puede describir el proceso en términos sencillos en tal forma que aun un niño (el cual será su aparato) pueda seguirlo? Piense en esto.

Primero, debemos suponer que el niño está familiarizado con el alfabeto. Segundo, debemos suponer que el niño tiene un conocimiento suficiente del idioma inglés para entender las instrucciones que le damos, y que es lo suficientemente inteligente para ejecutar dichas instrucciones correctamente. (Esto corresponde al *lenguaje* asociado al aparato y a las *operaciones* que éste es capaz de ejecutar.) Podemos llegar entonces a algo como esto:

Una forma de buscar una palabra en un diccionario palabra por palabra y página por página, consiste en recorrerlo desde el principio hasta que usted encuentra la palabra que busca o llegar al final del diccionario sin haberla encontrado. Esto, en esencia, es un *método de solución*. Podemos trasformar en *procedimiento* este *método de solución*, de la siguiente manera:

Compare la palabra dada con la primera palabra del diccionario; si son iguales, acabe la búsqueda y lea el significado de la palabra dado por el diccionario. Si esto no es así, compare con la segunda palabra del diccionario. . . , y así sucesivamente con cada palabra del diccionario. Si llega al final del diccionario sin encontrar la palabra, entonces abando-

* Usamos el término *método de solución* como un término impreciso que corresponde al término preciso de *procedimiento*.

ne e informe sobre el fracaso de encontrarle un significado a la palabra buscada.

Este es un procedimiento completamente válido pero consume mucho tiempo. Lo que debemos hacer para acelerar el proceso es pasar sobre grandes cantidades de palabras sin detenernos a compararlas individualmente con la palabra cuyo significado buscamos. Con el fin de evitar pasar por alto la palabra que buscamos, es necesario tener un poco de cuidado al definir cuándo es seguro dicho salto. Entonces sugerimos el siguiente procedimiento alternativo:

Considere cada letra de la palabra dada en su orden, comenzando con la letra inicial de la palabra y con el diccionario abierto en la primera página.

Para cada letra, en su orden, salte hacia adelante a partir del punto que haya alcanzado en el diccionario, hasta que llegue o bien a una palabra que tenga la misma letra en la misma posición de la palabra dada, o a una palabra que tenga en dicha posición una letra que esté más adelantada en el alfabeto. En el último caso, retroceda a la palabra a partir de la cual usted ha saltado, y trate de saltar en pasos más y más pequeños, verificando, finalmente las palabras en forma individual. Si llega aún a una palabra que tiene una letra más lejana en el alfabeto que la palabra buscada, puede sacar como conclusión que la palabra dada no está en el diccionario. De lo contrario, continúe con la letra siguiente de la palabra buscada y repita el proceso. O bien usted se detiene cuando llegue a la palabra dada y lee su significado, o bien se detiene cuando se le acaben las letras de la palabra sin haberla encontrado en el diccionario.

El procedimiento se puede mejorar, pero el que hemos dado puede servirle para darle alguna idea del grado de cuidado necesario para especificar un procedimiento. ■

Discusión
*

El siguiente ejercicio le da la oportunidad de ensayar la construcción de un procedimiento. El problema que le sirve de base es muy sencillo; usted puede encontrarlo en cualquier test de inteligencia. Dadas tres barras rígidas, usted puede colocarlas en forma de triángulo —o bien declarar la tarea imposible. Pero usted haría esto manipulando las barras mismas y trabajaría al azar, usando ambas manos simultáneamente, en lugar de hacerlo en forma deliberada, etapa por etapa, lo cual caracterizaría a un procedimiento. Además, los computadores no están diseñados para manipular barras, así que, usted tendrá que inventarse alguna otra forma de encarar el problema si (como es bueno hacerlo) piensa acerca de su procedimiento como si fuera especificado para una máquina (y, en consecuencia, a la cual no se le piden ni inteligencia ni capacidad de raciocinio para ejecutar el problema).

Ejercicio 1

Ejercicio 1
(5 minutos)

Dadas las longitudes de tres barras rígidas, diseñe un procedimiento para decidir si ellas forman o no los lados de un triángulo. ■

8.2.2 Definición formal de algoritmo

8.2.2

Tema principal
* *

Un *algoritmo* es un procedimiento que también posee las siguientes características:

(i) Se obtiene una solución del problema, si esta existe, después de haberse ejecutado un número *finito* de etapas discretas. En otras palabras, la ejecución del procedimiento finaliza si existe una solución.

Debe entenderse que el número de etapas ejecutadas, aunque finitas, puede ser muy grande y que, además, la ejecución puede ser un proceso largo y complicado si el aparato que ejecuta dicho procedimiento es complicado.

(ii) El procedimiento, cuando se ejecuta, resuelve también, inicial o concurrentemente, el problema relacionado: "¿Tiene el problema dado una solución?"

Lo anterior nos lleva a definir un algoritmo de la siguiente manera: un procedimiento para la resolución de un problema P, en un aparato D, es llamado un **algoritmo** para P en D, si después de la ejecución de un número *finito* de etapas, o bien se obtiene una solución para P (si esta existe) o D determina e informa que P no tiene solución.

Definición 1
* *

Ejemplo 1

Ejemplo 1

Supongamos que deseamos calcular el valor del entero positivo x que satisfaga la ecuación $a + x = b$ para enteros positivos dados a y b, utilizando un aparato que puede entender y ejecutar únicamente las siguientes instrucciones:

(i) Sumar dos enteros positivos para obtener su suma;
(ii) Comparar un par de enteros positivos y reconocer si son iguales o no.
(iii) Almacenar las cantidades involucradas en el cálculo.
(iv) Generar el siguiente de cualquier entero positivo en la sucesión habitual de los enteros.

Suponemos que el lenguaje comprensible para el aparato para el cual el algoritmo se especificará, es una versión modificada del idioma inglés y el cual es adecuado para expresar las operaciones sin ambigüedad. ■

Solución del ejemplo 1

Podríamos tratar de formular el algoritmo como sigue: verifique cada número sucesivo del conjunto de los enteros positivos, comenzando con el entero 1, aplicando las siguientes etapas:

Etapa 1 Forme la suma $s = a + i$, donde a es el entero positivo dado en la ecuación e i es el entero positivo que se está examinando.

Etapa 2 Compare el entero s con el entero b. Si son iguales, entonces tome el valor de i como el valor de la solución requerida, x: en caso contrario, continúe verificando con el siguiente entero i.

Estas reglas describen ciertamente un *procedimiento*, ¿pero describen un *algoritmo*? Por desgracia, no lo hacen. (Considere la aplicación de estas reglas a la ecuación $7 + x = 4$.) Si la ecuación no tiene solución dentro del conjunto de los enteros positivos, entonces el aparato continuará verificando indefinidamente.

Esta dificultad se puede evitar si definimos de nuevo las reglas de la siguiente manera:

Etapa 0 Compare los valores de a y b; si son iguales, detenga la ejecución e informe que no hay solución: en caso contrario continúe con la etapa 1 que sigue, haciendo $i = 1$.

Etapa 1 Forme la suma $a + i$ para encontrar s, y la suma $b + i$ para obtener t, siendo i el entero positivo que se está verificando.

Etapa 2 Compare los valores de s y de b; si son iguales, tome entonces el valor de i como el valor de la solución requerida, x: en caso contrario continúe verificando en la etapa 3 que sigue.

Etapa 3 Compare el valor de t y el valor de a; si son iguales, termina la ejecución con no solución: en caso contrario continúe con la etapa 1 anterior, con el siguiente valor de i.

Este procedimiento es ahora un algoritmo también, porque si hay una solución entera de la ecuación $a + x = b$, el algoritmo la encontrará después de un número *finito* de etapas (la encontrará reconociendo la verificación $s = b$); si no hay tal solución, esto será descubierto en un número finito de etapas o en la primera etapa o cuando la comparación $t = a$ se satisface. ■

Ejercicio 1

Ejercicio 1
(2 minutos)

Sin introducir el concepto de enteros negativos ni las técnicas para manipular ecuaciones, ¿puede usted cambiar el método de solución dado en el ejemplo 1 en tal forma que resulte menos laborioso que el algoritmo dado? ■

SUGERENCIA: usted tendría que modificar o agregar instrucciones a las instrucciones que suponemos el aparato puede entender y ejecutar.

Hemos definido un algoritmo como un procedimiento que capacita a un aparato para evaluar la solución de un problema o para determinar que ninguna solución existe en un número finito de pasos. Deseamos finalizar esta sección describiendo un método de clasificación de algoritmos de acuerdo con el número de etapas que comprenden. Podemos identificar los siguientes casos:

(i) Algoritmos en los cuales el número de etapas es fijo, independientemente de los datos o en los que el número de etapas no sobrepasa algún máximo absoluto (llamado cota superior).

(ii) Algoritmos para los cuales el número de etapas está relacionado a los datos con los que opera el algoritmo, en tal forma que podemos calcular por consideraciones a priori o el número de etapas comprendidas o bien una cota superior de las mismas.

(iii) Algoritmos para los cuales el número de etapas depende del problema, en tal forma que no podemos predecir ninguna cota superior para el número de etapas comprendidas.

Un ejemplo de algoritmo del primer tipo podría ser un algoritmo que convirtiera a un símbolo único en un código buscando en un diccionario de símbolos. Por ejemplo, convirtiendo una de las letras $A, B, C, \ldots, Z$ a su entero correspondiente $1, 2, 3, \ldots, 26$ buscando en el alfabeto normal. En este caso, 26 es una cota superior.

El ejemplo 1 (discutido anteriormente) es un ejemplo de algoritmo del segundo tipo. Dados los coeficientes en la ecuación, podemos calcular el número de etapas empleadas.

Los algoritmos del tercer tipo pueden presentarse cuando investigamos en un conjunto infinito. Por ejemplo, investigando en el conjunto de los

números enteros por el número primo* más próximo a 2^n para algún valor dado de n. Otro ejemplo podría ser un algoritmo de cálculo concerniente a una sucesión convergente (por ejemplo, el procedimiento iterativo para resolver la ecuación $f(x) = 0$, descrito en la *Unidad 2*). En este caso tenemos un criterio que debe ser satisfecho por cualquier resultado aceptable, y los cálculos continúan hasta que el criterio sea satisfecho. En ambos casos, suponiendo que la aritmética sea ejecutada exactamente, sabemos que el proceso debe terminar después de un número finito de etapas. De hecho, debemos estar en capacidad de *demostrar* la finalización con el fin de asegurarnos de que tenemos un algoritmo y no solo un procedimiento. Aun en dichas circunstancias, a menudo no seremos capaces de predecir el número de etapas necesario para producir un resultado.

La idea de que la precisión de un resultado se aumenta con el número de pasos ejecutados sugiere que con la ejecución de un número arbitrariamente grande de etapas cualquier grado de precisión se puede alcanzar. Sin embargo, como hemos visto, (en la *Unidad 2*) cuando se calcula con un número fijo de cifras decimales, la acumulación de los errores de redondeamiento restringe la precisión que se puede alcanzar. En consecuencia, puede ser que nunca lleguemos a satisfacer la condición de finalización a pesar del hecho de que teóricamente podamos probar la convergencia.

Por esta razón los algoritmos del tipo (ii) se convierten, a menudo, en algoritmos del tipo (i) imponiendo un límite arbitrario al número de etapas ejecutadas. En este caso trasformamos el problema de

> "Encontrar una solución de... si alguna existe"

a

> "Encontrar una solución de... en menos de N etapas, si alguna existe".

Ejercicio 2

Ejercicio 2
(5 minutos)

Formular un algoritmo del tipo (ii) para decidir si un entero positivo dado, $p > 2$, es primo. Especificar las instrucciones que usted supone el aparato debe ser capaz de entender e implementarlas con el fin de llevar a cabo su algoritmo. ■

Ejercicio 3

Ejercicio 3
(3 minutos)

Formular un algoritmo del tipo (iii) para calcular $\sqrt{2}$ con un número arbitrario de cifras decimales en un aparato capaz de ejecutar la suma y la división. (Véase la Sección 7.1.2 de la *Unidad 7, Sucesiones y límites I.*) ■

Llegamos ahora al punto central de la automatización en la solución de problemas, a saber: "¿Qué problemas se pueden resolver por medio de algoritmos?" Desde el punto de vista intuitivo la respuesta ha sido ofrecida por las definiciones formales dadas anteriormente: "Cualquier problema para el cual podamos especificar un método sistemático, etapa por etapa, de la evaluación de una solución en un número finito de

* Un número primo es un entero $p > 1$ con la propiedad de que si a y $\frac{p}{a}$ son enteros, entonces $a = \pm 1$, o, $a = \pm p$. Por ejemplo, 1, 2, 3, 5, 7, 11, 13 son números primos.

(*continúa en la página 11*)

Solución 8.2.1.1

Solución 8.2.1.1

El procedimiento enunciado a continuación depende del siguiente sencillo teorema de geometría plana:

La suma de las longitudes de cualesquier dos lados de un triángulo es mayor que la longitud del tercer lado.

Sea a, b, c las longitudes de las tres barras. El procedimiento puede ser enunciado en tres etapas como sigue:

Etapa 1 Formar la suma $a + b$ y denotar el resultado por s. Comparar a s con c. *Si* s es menor o igual a c, deducimos que las barras no pueden formar un triángulo y el procedimiento es *completo*. De lo contrario:

Etapa 2 Formar la suma $a + c$ y denotar el resultado por t. Comparar t con b. *Si* t es menor o igual a b, concluimos que las barras no pueden formar un triángulo y el procedimiento es *completo*. De lo contrario:

Etapa 3 Formar la suma $b + c$ y denotar el resultado por u. Comparar u con a. *Si* u es menor que a, concluimos que las barras no pueden formar un triángulo y el procedimiento es *completo*. De lo contrario, concluimos que las barras pueden formar los lados de un triángulo y el procedimiento es *completo*.

■

Solución 1

Solución 1

Se alcanza una simplificación considerable cuando se comparan los valores dados de a y b, primero.

Etapa 1 Compara a con b. Si b es mayor que a, continuar con la etapa 2: en caso contrario, parar la ejecución —la ecuación no tiene solución positiva entera.

Etapa 2 Hacer $i = 1$.

Etapa 3 Formar la suma $s = a + i$.

Etapa 4 Comparar s con b. Si son iguales, detener la ejecución —la solución es $x = i$: de lo contrario, volver a la ejecución de la etapa 3 con el siguiente valor de i.

La primera etapa de este algoritmo requiere de este aparato, que sea capaz de decir si un número es mayor que otro, lo cual es más que pedirle que sea capaz de decir cuándo dos números son iguales o no. Una modificación conveniente de la instrucción (ii) del ejemplo 1 arreglará este problema.

■

Solución 2

Solución 2

Esperamos que el aparato sea capaz de entender e implementar las siguientes instrucciones:

I Dados dos enteros a y b, determinar si a es o no mayor que b.
II Dado un número positivo a, extraer la raíz cuadrada de a correctamente con una cifra decimal.
III Dados dos números a y b, formar el cociente a/b.
IV Dado un número a, determinar si a es o no un número entero.
V Seleccionar el siguiente entero de la sucesión: 1, 2,. . .
VI Almacenar la información.

Podemos construir el algoritmo del tipo (ii) como sigue:

Etapa 1 Dado p, determinar $M = \sqrt{p}$ con una cifra decimal y almacenar a M.

Etapa 2 Hacer $m = 2$.

Etapa 3 Formar el cociente p/m.

Etapa 4 Si este cociente es un número entero, informar que p no es primo y detenerse; en caso contrario continuar con la etapa 5 que sigue.

Etapa 5 Comparar m y M. Si $m > M$, informar que p es primo y detenerse: de lo contrario, siga con la etapa 6.

Etapa 6 Seleccionar el siguiente entero positivo que sigue al actual valor de m y dé a m este nuevo valor; vuelva a la etapa 3.

Obsérvese que, si deseamos reducir las capacidades requeridas del aparato, podemos hacerlo de dos maneras:

Primeramente, es innecesario que el aparato sea capaz de extraer raíces cuadradas. Si una parte de los datos es un número N tal que $N^2 \geqslant p$, entonces N se puede suministrar como dato al aparato y utilizarse en lugar de M.

En segundo lugar, podemos abolir el requerimiento de que dados a y b (enteros positivos), el aparato pueda determinar si $a > b$. En efecto, podemos pedir al aparato que forme todos los cocientes p/m a medida que m varía desde 1 hasta $p - 1$, y determine si cada cociente es o no un número entero.

Por otra parte, si exigimos que el aparato tenga más capacidades, podríamos requerir que pueda reconocer a todos los números primos menores que p. Entonces, sencillamente tiene que dividir a p por todos esos números y verificar los cocientes correspondientes. ■

(*viene de la página 9*)

pasos". Sin embargo, nuestras definiciones especifican que el algoritmo debe ser descrito en un lenguaje que pueda ser interpretado por el aparato y especificado en función de las instrucciones que éste pueda ejecutar. En la siguiente sección consideraremos los problemas que se presentan cuando tratamos de especificar formalmente un algoritmo, y continuaremos hasta considerar las instrucciones disponibles si el aparato ejecutor del algoritmo es un computador digital moderno.

8.2.3 Descripción de un algoritmo por medio de un diagrama de flujos

8.2.3

Introducción

La descripción de un algoritmo requiere:

(i) que tengamos a nuestra disposición un *lenguaje* para describir el algoritmo y el cual sea aceptable por el aparato que va a ejecutar el algoritmo,

y

(ii) que seamos capaces de especificar un número finito de etapas, en función de las instrucciones que el aparato pueda ejecutar, y de

(*continúa en la página 12*)

Solución 8.2.2.3

Solución 8.2.2.3

La sucesión dada en la Sección 7.1.2 de la *Unidad 7* (Método de Newton para la extracción de raíces cuadradas) es conveniente:

$$x_0 = 1 \quad \text{(digamos)}$$

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{2}{x_n}\right) = \frac{x_n}{2} + \frac{1}{x_n}$$

o, en otras palabras, comenzar con una suposición inicial, $x = 1$; y sucesivamente remplazar x_{antiguo} por el valor de x_{nuevo} calculado a partir de la fórmula:

$$x_{\text{nuevo}} = \tfrac{1}{2}x_{\text{antiguo}} + \frac{1}{x_{\text{antiguo}}}$$

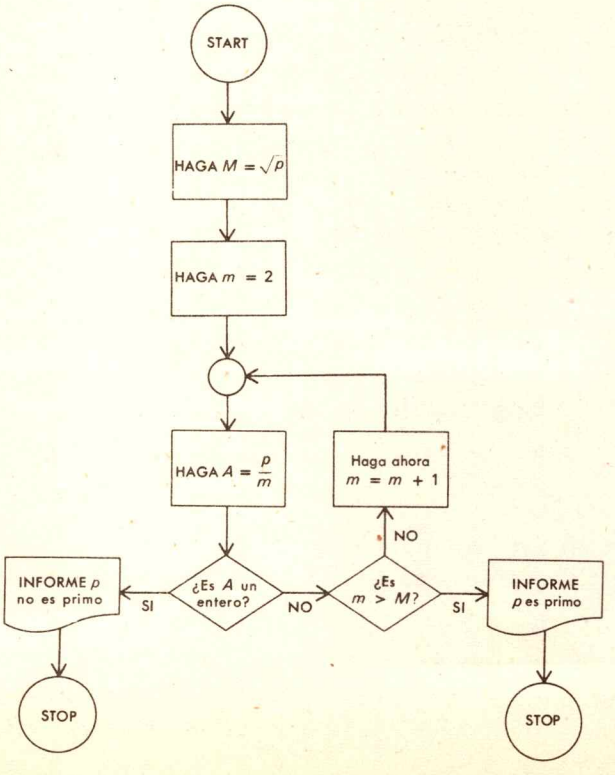
Si se ha propuesto calcular $\sqrt{2}$ con un número arbitrario de cifras decimales, entonces el aparato que ejecuta el algoritmo debe ser capaz de llevar a cabo operaciones con números arbitrariamente grandes. ■

(*viene de la página 11*)

ordenar esas etapas en sucesión, en tal forma que la ejecución de las instrucciones conduzca a la solución después de que éstas hayan sido procesadas en un número finito de etapas.

Las etapas ordenadas en sucesión pueden ser consideradas como la estructura del algoritmo. Una forma conveniente de comunicar un algoritmo de una persona a otra, cuando el algoritmo es "razonablemente" sencillo, es usando lo que se conoce como un **diagrama de flujos**. Este es una representación pictórica del algoritmo en la cual las diferentes etapas que se van a ejecutar, se indican en recuadros con instrucciones específicas dentro de ellos, y el orden en el cual las etapas se han de ejecutar se indica por flechas que unen a dichos recuadros.

Definición 1
* *



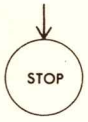
Como ejemplo, consideremos un algoritmo que usted ya ha visto, a saber, el descrito en la solución del ejercicio 8.2.2.2 ("¿es p primo?"), y expresado por medio de un diagrama de flujos. Aunque usted no haya visto antes ninguna clase de diagramas de flujos, encontrará bastante fácil seguir éste, pues el diagrama de flujos describe claramente al algoritmo y lo hace sin ambigüedad.

Tema principal
* *

La utilidad de un diagrama de flujos para la descripción de un algoritmo se incrementa cuando cada nodo (recuadro) del diagrama se conforma de tal manera que especifica la naturaleza de la instrucción que se debe ejecutar en el nodo en consideración. Las formas son diseñadas para dar de una sola mirada la naturaleza de las instrucciones. No hay ningún conjunto estándar de formas internacionalmente aceptado para estos recuadros, aunque existe un cierto número de formas aceptadas nacionalmente. Las formas presentadas a continuación representan las más comúnmente aceptadas en la práctica. Las instrucciones puestas dentro de los recuadros del diagrama de flujos se expresan en inglés y en la notación usual de las matemáticas y la lógica y son simplemente ilustrativas.

Comienzo y terminación

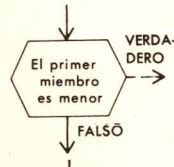
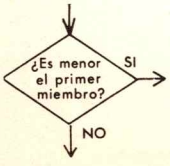
Notación
* *

Símbolo	*Interpretación*
	Comienzo del algoritmo (esto es, el punto en el cual la ejecución comienza).
	Final del algoritmo (esto es, el punto en el cual la ejecución termina).

Procesamiento

Símbolo	*Interpretación*
	Una etapa que se debe ejecutar, está representada por un recuadro rectangular. Si es necesaria mayor claridad, los procesos que se deben llevar a cabo durante la ejecución de una simple etapa se pueden subdividir en subprocesos especificados en varios recuadros. Además, el texto que especifica el proceso se puede ampliar por el uso de notas marginales.

Decisiones binarias

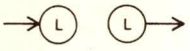
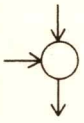
Símbolo	*Interpretación*
	Las dos salidas del recuadro están marcadas VERDADERO y FALSO, y la apropiada es escogida de acuerdo con la afirmación contenida en el recuadro. En consecuencia, escogemos una de dos vías de acción que verifica la afirmación.
	Las dos salidas del recuadro están marcadas SI y NO y la salida apropiada se toma de acuerdo con la respuesta dada a la pregunta contenida en el recuadro.

Notas

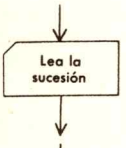
(i) Es claro que los dos últimos símbolos son intercambiables, en el sentido de que una afirmación puede ser puesta como pregunta y viceversa. La escogencia de la forma del recuadro que se debe utilizar depende de la evaluación individual acerca de la forma "natural" de representar una decisión particular.

(ii) Los procesos decisorios más complicados, se deben subdividir en un conjunto de decisiones binarias, esto es, decisiones entre dos posibilidades como ha sido expuesto.

Conexiones

Símbolo	*Interpretación*
	Cuando no es posible continuar un diagrama de flujos inmediatamente después del recuadro precedente, entonces se utiliza un círculo conectivo *marcado*. Dos o más de tales símbolos representan el mismo punto en el diagrama de flujos, esto es, el mismo estado en la ejecución del algoritmo.
	Cuando dos o más líneas de flujos se encuentran (confluyen), la unión que forman se denota por un círculo conectivo. En este caso la marca se puede omitir, a menos que la unión también sirva como conexión en el sentido expresado anteriormente.

Comunicación

Símbolo	*Interpretación*
	La información que va a servir como datos de entrada al algoritmo está especificada en un recuadro de entrada.
	La información que forma el resultado suministrado por el algoritmo está especificada en un recuadro de salida.

Como el hombre común tiene considerables poderes de raciocinio, los diagramas de flujos pueden representarse informalmente en una amplia variedad de formas y ser aun correctamente interpretados. Para nuestros propósitos, es más satisfactorio restringirnos a la notación formal especificada anteriormente, y esto nos ayudará a acostumbrarnos a definir un algoritmo en una forma conveniente para ser ejecutada por un computador.

Como primera ilustración sobre el uso de los diagramas de flujos consideremos el sencillo algoritmo de convertir un entero decimal a la notación binaria. Comenzamos por especificar el siguiente algoritmo en función de las instrucciones que un aparato automático pueda entender y ejecutar. En consecuencia, algunas de las instrucciones son más formales que las requeridas para la comunicación humana.

Ejemplo 1

Ejemplo 1

(Véase RB 12)

Describir por medio de un diagrama de flujos un algoritmo que convierta a un entero decimal dado en su forma binaria. ■

Solución del ejemplo 1

Supongamos que el entero dado es n, y que la respuesta requerida es el número binario

$$b_k b_{k-1} \ldots b_2 b_1 b_0$$

donde cada $b_i = 0$, ó 1 (para $i = 0, \ldots, k$, $k \in Z^+$). Entonces tendremos

$$n = 2^k b_k + 2^{k-1} b_{k-1} + \cdots + 2^2 b_2 + 2b_1 + b_0$$

de lo cual es evidente que b_0 es el residuo cuando n es dividido por 2. Si ponemos

$$n = 2q_1 + b_0$$

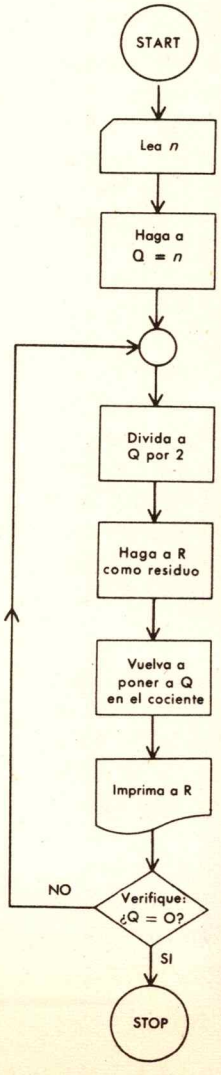
entonces el cociente es

$$q_1 = 2^{k-1} b_k + 2^{k-2} b_{k-1} + \cdots + 2b_2 + b_1$$

y en consecuencia b_1 es el residuo cuando q_1 se divide por 2 y así sucesivamente... En general, tendremos

$$q_i = 2q_{i+1} + b_i$$

Comenzando con $q_0 = n$, podemos obtener los b_i sucesivos como los residuos de la división de los cocientes sucesivos q_i por 2. Entonces este será nuestro algoritmo para la conversión binaria y podemos expresarlo por medio de un diagrama de flujos, como sigue:



Usted debe haber observado que nuestro algoritmo imprime al número binario en el orden inverso.

$$b_0 b_1 b_2 \ldots b_{k-1} b_k$$ ■

Este hecho nos sugiere el siguiente ejercicio.

Ejercicio 1

Ejercicio 1
(5 minutos)

Dibujar un diagrama de flujos para un algoritmo que acepte como datos de entrada a cualquier entero decimal y dé como datos de salida al número binario correspondiente con sus dígitos colocados en el orden correcto. ■

Ejemplo 2

Ejemplo 2

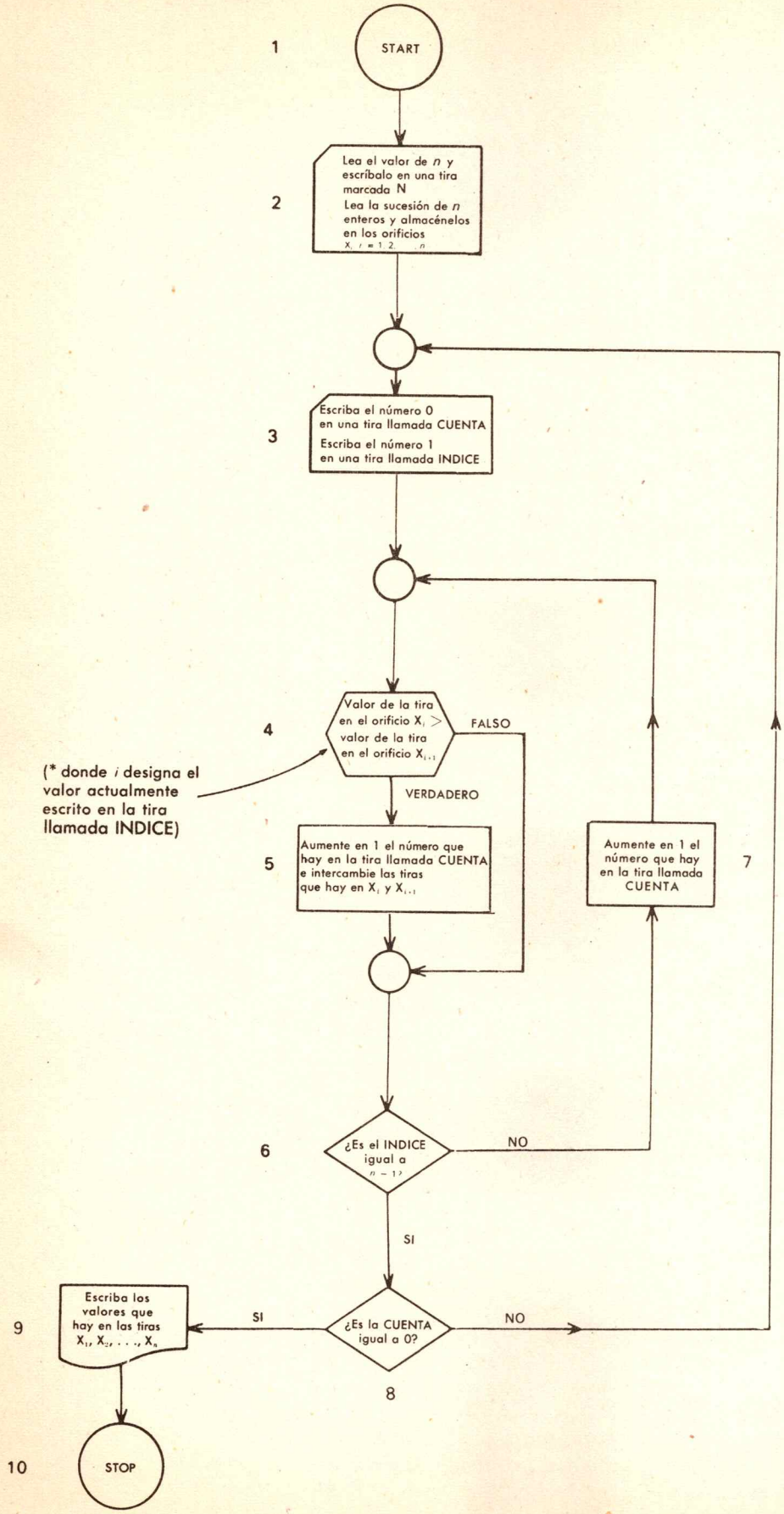
Dado como entrada un solo entero positivo n y una sucesión de n valores enteros diferentes, dibujar un diagrama de flujos para un algoritmo que dé como salida a los n enteros colocados en orden ascendente. ■

Solución del ejemplo 2

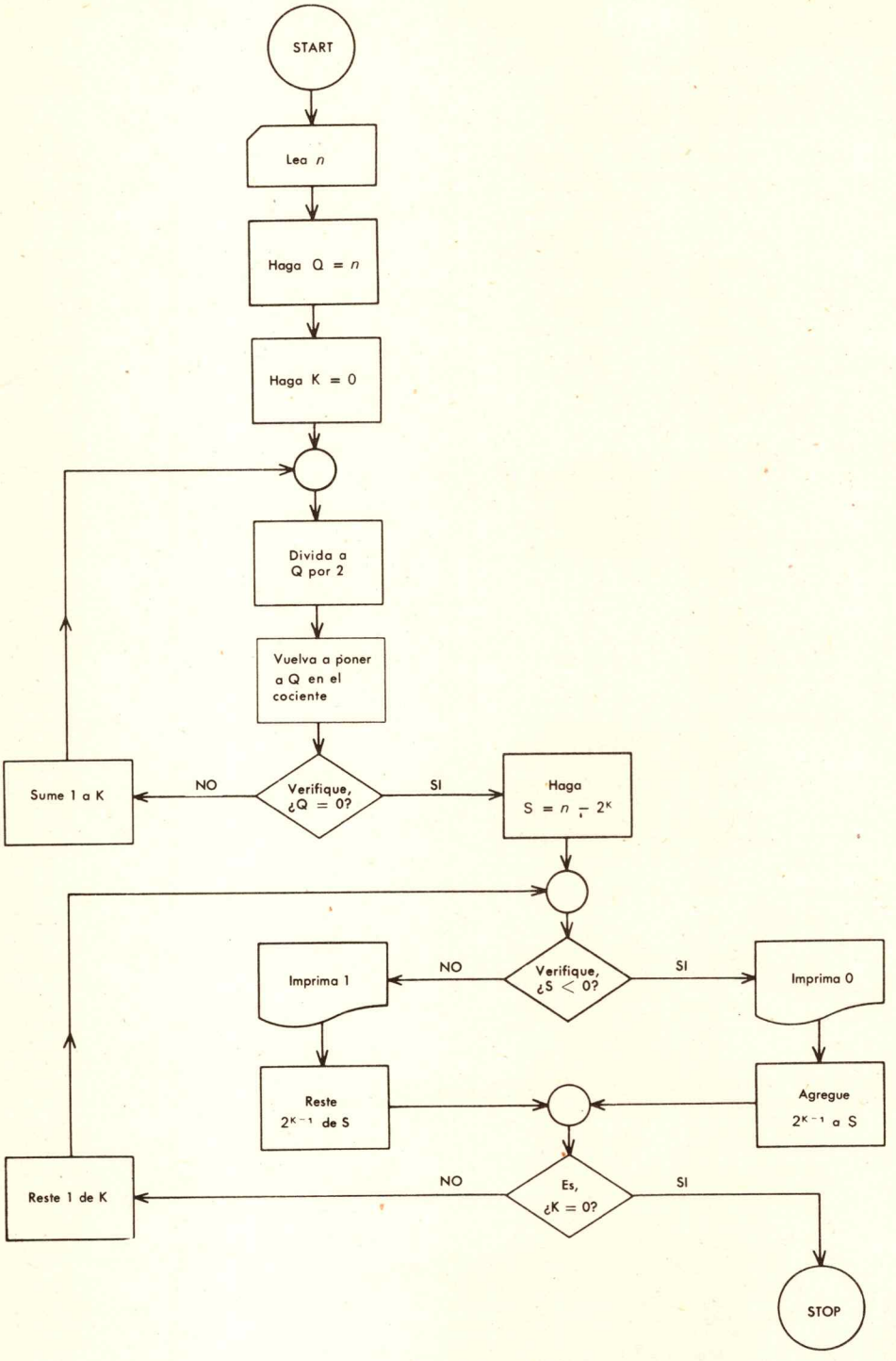
Nos es útil crear un modelo conceptual sencillo que nos ayude a visualizar el proceso que deseamos ejecutar. Supongamos que disponemos de un conjunto de más de n orificios marcados $X_1, X_2, X_3, \ldots$, capaces de contener cada uno, una tira de papel en la cual hay un número escrito, y un bloque de papel que nos suministra las tiras para escribir los números y en el cual podemos almacenar temporalmente la información. Llamaremos X_i al número almacenado en x_i.

Se presenta un algoritmo sencillo en el siguiente diagrama de flujos. En esencia, el método adoptado consiste en revisar los orificios $X_1, X_2, X_3, \ldots$, repetidamente. En cada revisión, comparamos los números que hay en los orificios consecutivos: si encontramos que X_i, es mayor que x_{i+1}, intercambiamos el contenido de X_i y X_{i+1}. Se conserva una cuenta del número de intercambios efectuados al final de cada revisión completa. Cuando encontremos un conteo igual a cero al final de una revisión completa, sabemos que no ha habido intercambios en dicha revisión; en consecuencia, tendremos $x_i \leqslant x_{i+1}$ para todo par de orificios adyacentes X_i, X_{i+1} y, por tanto, el contenido de los orificios está en orden ascendente.

Si usted pensó por sí mismo en la solución de este ejemplo, entonces habrá llegado probablemente a describir un diagrama de flujos diferente. Puede verificar su respuesta, o comprobar el diagrama de flujos dado anteriormente, *siguiendo* la ejecución del algoritmo, como sigue: si cada etapa del algoritmo se identifica marcando el correspondiente recuadro con los números 1, 2, 3,... (observe que los recuadros de conexión y convergencia *no* están marcados pues no constituyen etapas separadas del proceso), entonces el algoritmo se puede seguir por medio de una tabla como la que se presenta más adelante. Con propósitos ilustrativos, tomamos $n = 5$ y trabajamos con la secuencia 9, 7, 3, 4, 6.



(*continúa en la página 19*)



Claro está, hay otras soluciones a este ejercicio.

Si tiene tiempo, trate de extender el algoritmo para que incluya a los números no enteros. Podría considerar también el problema inverso, es decir, convertir números binarios en números decimales. ■

(*viene de la página 17*)

Etapa Completada	Valor de *N*	Valor de la CUENTA	Valor del INDICE	Valores contenidos en $X_1, X_2, \ldots, X_n$	Condición del test
1					
2	5			9, 7, 3, 4, 6	
3	–	0	1	–	
4	–	–	–	–	VERDADERO
5	–	1	–	7, 9, 3, 4, 6	
6	–	–	–	–	NO
7	–	–	2	–	
4	–	–	–	–	VERDADERO
5	–	2	–	7, 3, 9, 4, 6	
6	–	–	–	–	NO
7	–	–	3	–	
4	–	–	–	–	VERDADERO
5	–	3	–	7, 3, 4, 9, 6	
6	–	–	–	–	NO
7	–	–	4	–	
4	–	–	–	–	VERDADERO
5	–	4	–	7, 3, 4, 6, 9	
6	–	–	–	–	SI
8	–	–	–	–	NO
3	–	0	1	–	

Una sucesión semejante de intercambios dará como resultado la sucesión 3, 4, 6, 7, 9; la revisión subsiguiente no presentará intercambios (así el valor del conteo permanece igual a cero); en consecuencia, la pregunta en el recuadro decisorio de la etapa 8 nos da la respuesta SI; la sucesión ordenada se escribe, y el proceso termina.

Nota

Una convención útil de adoptar cuando se está especificando un valor en una tabla de seguimiento es escribir el valor en forma explícita únicamente cuando cambia: esto ayuda a concentrar la atención sobre

aquellos valores que cambian en una etapa particular. También es útil distinguir en cada etapa, entre valores indefinidos (por ejemplo valores aún no especificados) y valores que no han cambiado. En la tabla anterior se indica un valor indefinido por una posición en blanco y un valor que no ha cambiado por un guion. ■

Ejercicio 2

Ejercicio 2
(10 minutos)

Dado como entrada un número positivo n y un conjunto de n números enteros, dibuje un diagrama de flujos para un algoritmo que imprima como salida el valor del máximo número del conjunto y el miembro del conjunto que tenga mayor valor absoluto. ■

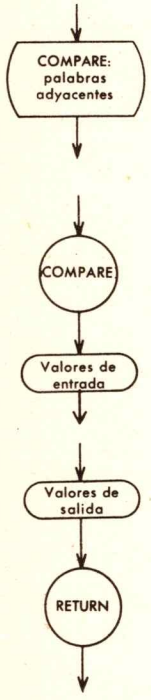
Discusión
* *

Los ejemplos y el ejercicio indican el nivel de detalle que se debe especificar para la ejecución de trabajos relativamente sencillos, si cada recuadro del diagrama de flujos contiene una instrucción (o pregunta). Si continuamos de esta manera, entonces los diagramas de flujos que especifiquen algoritmos de trabajos más complejos, se harán más complicados y esto reducirá la utilidad. Con el fin de evitar esto, podemos extender la notación utilizada en el diagrama, incluyendo un símbolo que pueda "resumir" las etapas contenidas en un número de recuadros diferentes. Esta técnica corresponde al diseño de un diagrama de flujos correspondiente al de una máquina hipotética capaz de ejecutar operaciones más complejas que las que hemos expuesto hasta ahora. Un ejemplo de tal extensión es el concepto de subrutina.

Notación
* *

Subrutinas

Símbolo



Interpretación

Definición 2
* *

El símbolo de subrutina se utiliza para indicar una subrutina, esto es, una etapa larga de un programa la cual se discute y diagrama detalladamente en alguna otra parte. El texto escrito en el recuadro especifica el nombre de la subrutina y la información sobre la cual opera.

Estos símbolos se utilizan para indicar el *comienzo* y el *final* de la descripción, en el diagrama de flujos, de la subrutina (en lugar de START, STOP). El recuadro de comienzo contiene el nombre de la subrutina: el recuadro de finalización contiene la instrucción RETURN. Los recuadros en forma oblonga especifican los valores de entrada y de salida asociados con la ejecución de la subrutina.

Ejemplo 3

Ejemplo 3

Dibujar un diagrama de flujos para una subrutina que acepte como datos de entrada dos palabras (formadas por una sucesión de símbolos alfabéticos) almacenadas en un aparato en las direcciones identificadas por las letras A y B, y un entero T. La subrutina debe dar como resultado las dos palabras almacenadas en el orden del diccionario, en las

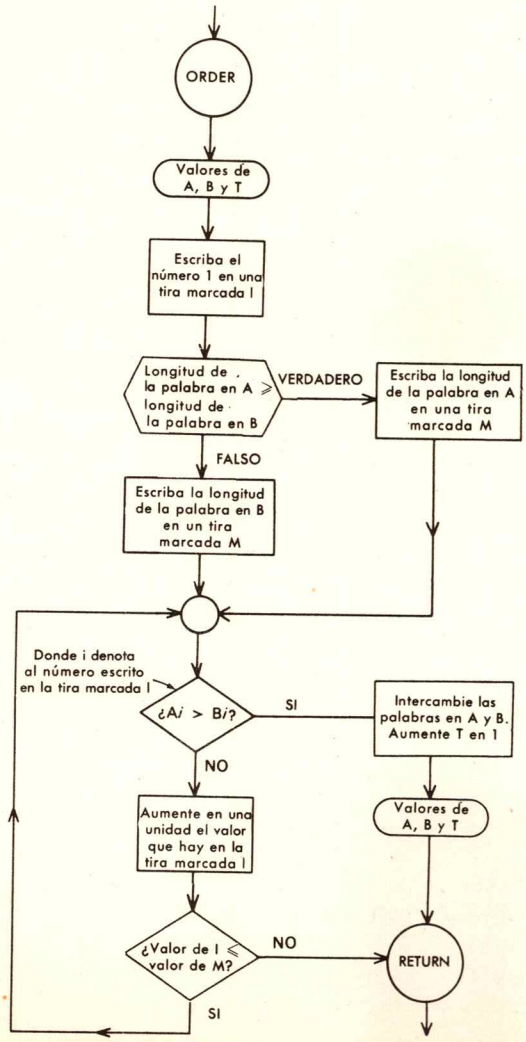
direcciones A y B, y un entero que es, o bien T, o T aumentado en una unidad si los contenidos de A y B han sido intercambiados. Suponga que el aparato que ejecuta el algoritmo es capaz únicamente de comparar un par de símbolos simultáneamente, usando las relaciones "igual que", "menor que", y "mayor que" (los cuales usamos en lugar de "es lo mismo que", "se presenta en el alfabeto antes de" y "se presenta en el alfabeto después de" respectivamente). ■

Solución del ejemplo 3

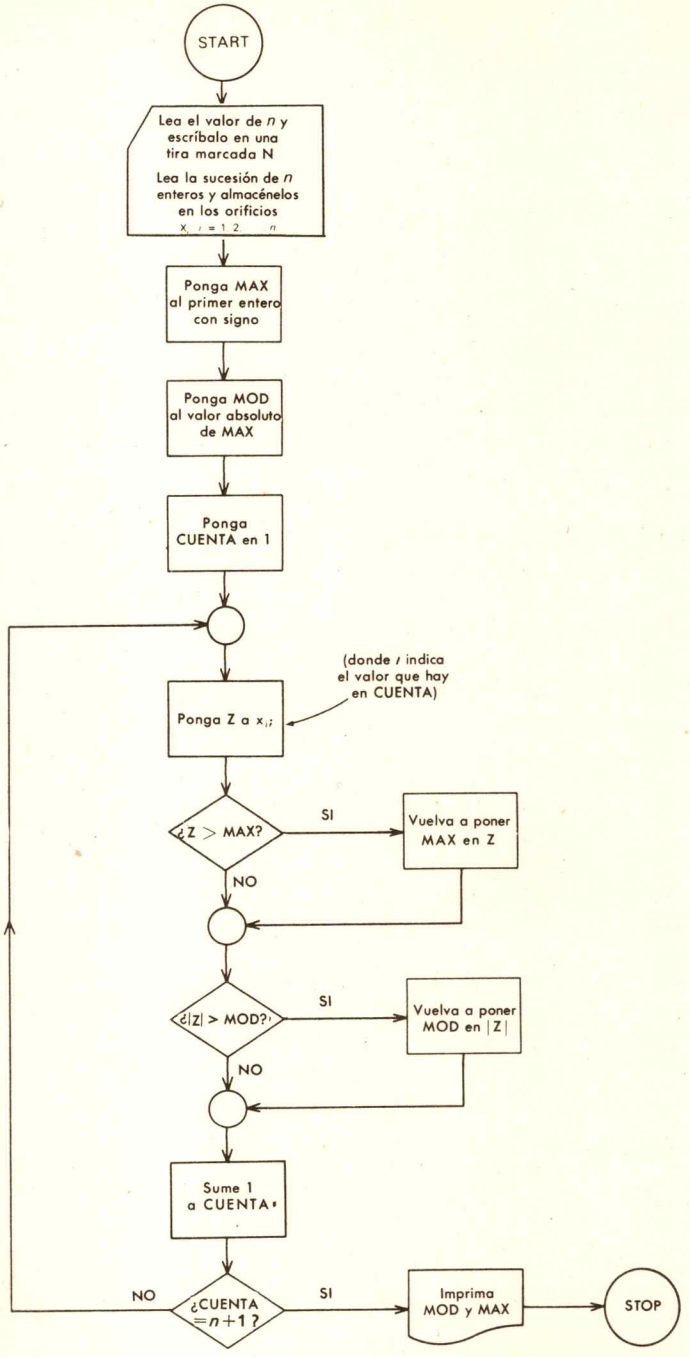
Usaremos A_i para indicar la i-ésima letra de la palabra situada en la dirección A, y similarmente B_j para indicar la j-ésima letra de la palabra situada en la dirección B. Suponemos que cada palabra está almacenada comenzando a la izquierda de cada dirección y que hay suficientes posiciones en A y en B para almacenar cada palabra. Cualquier posición no ocupada a la derecha de la palabra queda en blanco y convenimos en considerar un blanco como "menor que" cualquier símbolo alfabético. Con estas convenciones, las palabras COMPUTER y COMPUTING, por ejemplo, quedarán almacenadas en las direcciones A y B, como sigue:

A	C	O	M	P	U	T	E	R			B	C	O	M	P	U	T	I	N	G	
i =	1	2	3	·	·	·	·	8	9	10	j =	1	2	·	·	·	·	·	·	9	10

Utilizando este modelo, un diagrama de flujos conveniente, se puede diseñar así:



(*continúa en la página 22*)



■

(*viene de la página 21*)

Observe que los recuadros oblongos se usan solamente antes de un recuadro marcado RETURN, para especificar valores que o bien han sido colocados por la subrutina o han sido cambiados por ésta. En consecuencia, cuando ningún intercambio se presenta, no se especifica ninguna salida. ■

8.3 CONCEPTO DE MAQUINA LOGICA

8.3

8.3.0 Introducción

8.3.0

Introducción
* *

Al comienzo de la Sección 8.2, dijimos que concentrábamos nuestra atención sobre las "técnicas para concebir un plan de solución en tal forma que éste pueda ser formalmente especificado en función de un conjunto de instrucciones que una máquina lógica pueda interpretar". Llamábamos a esa actividad, "encontrar un *procedimiento* o *algoritmo*". Hacíamos notar que, si este objetivo se puede alcanzar, la fase siguiente a la actividad de resolver problemas (ejecutar el plan o el algoritmo) puede ser efectuada *automáticamente* por una máquina lógica. Estudiamos brevemente los problemas inherentes a la creación de la descripción formal de un algoritmo. En esta sección deseamos concentrarnos en los problemas que se presentan cuando una máquina lógica ejecuta un algoritmo. Con el fin de estudiar dichos problemas debemos crear un modelo conceptual sencillo y conveniente de máquina lógica. Comenzaremos por examinar un modelo de máquina cuya característica primordial es ser capaz de ejecutar una sucesión ordenada de instrucciones sencillas.

La máquina lógica empleada corrientemente para ejecutar algoritmos es el computador digital automático; en consecuencia, lo desarrollaremos en tal forma que llegará a ser el arquetipo de un computador moderno. Esto nos lleva a definir una notación concisa y sin ambigüedades, basada en la operación de nuestro modelo conceptual, la cual pueda ser utilizada para describir algoritmos ejecutables por un computador real. Estas nociones se presentan en relación con el proceso de especificación de un problema que se debe resolver, usando el sistema de computadores existente que forma la base del trabajo práctico asociado a la componente de computación del Curso Básico de Matemáticas.

El modelo de máquina lógica (sistema de computación) que se presenta, aunque extremadamente sencillo, es más que adecuado para conformar una base para el estudio de la ejecución de algoritmos y es, de hecho, compatible con la estructura de un computador digital moderno.

8.3.1 El concepto de variable: Elemento de almacenamiento

8.3.1

Tema principal
* * *

En la Sección 8.2, cuando discutíamos la ejecución de algoritmos para escoger números, introdujimos un modelo en el cual los números eran escritos en tiras de papel y éstas eran depositadas en orificios marcados. En esta sección generalizamos ligeramente este modelo con el fin de que nos dé el elemento de almacenamiento de nuestro modelo de computador.

Definición 1
* *

Un ítem de información (por ejemplo, un número o una sucesión de letras) manipulada (esto es, trasformada) por un algoritmo, se llama una variable. Asociada a cada variable tenemos un nombre, que nos sirve para identificar la variable, y un valor actual que especifica el valor de la variable en cualquier momento particular. Obsérvese que el valor de la variable puede *cambiar* de momento a momento como resultado de la ejecución del algoritmo con el cual está asociada. (El uso del término *variable* no debe ser confundido con el

uso que se le da en las matemáticas convencionales (*Ver la Unidad 1, Funciones*). Por ejemplo, la letra x en la especificación funcional

$$f: x \longmapsto x^2 \qquad (x \in R)$$

es una variable en el sentido matemático convencional; aquí tiene ciertamente un nombre (la letra x) y un valor asociado, pero este valor está determinado por la declaración subsecuente de las condiciones que x debe satisfacer).

En nuestro modelo conceptual, asociamos a cada variable un sitio único en el cual podamos almacenar el valor actual de la variable. Este sitio de almacenamiento se puede visualizar como un orificio en el cual colocamos una tira de papel donde está escrito el valor actual de la variable. Con el fin de identificar la variable supondremos que el orificio se puede marcar. La asignación de nombres a los orificios será un proceso dinámico, en tal forma que los sitios de almacenamiento puedan ser usados una y otra vez cuando ejecutan diferentes algoritmos. En consecuencia, imaginamos que el nombre está escrito en una cinta adhesiva y pegado al orificio que contiene el valor actual de la variable correspondiente. **Definición 2**

Estas ideas nos llevan a especificar formalmente al *elemento de almacenamiento* en nuestro modelo. Esta definición se puede resumir como sigue:

El elemento de almacenamiento en nuestro modelo consiste de una colección de sitios de almacenamiento (visualizados por orificios) para la cual el modelo puede recibir y ejecutar las siguientes instrucciones: **Definición 3** * *

(i) Dar un nombre a un sitio de almacenamiento no identificado, pegando una marca sobre un orificio. Este proceso se llama declarar una variable. Por el momento, la forma que pueda tomar un nombre se deja sin aclarar. **Definición 4** * *

(ii) Asignar un valor a una variable insertando en el sitio de almacenamiento apropiado, una tira de papel en la que está escrito el valor de la variable. Este valor *remplaza* a cualquier otro valor que pueda estar almacenado en ese sitio, de tal manera que, si la localización contiene ahora un valor, entonces éste es destruido cuando se le asigna al sitio otro valor nuevo. **Definición 5** * *

(iii) Copiar el valor de una variable leyendo el valor escrito en la tira contenida en el sitio de almacenamiento asociado a esta variable, y escribiéndolo en una tira de papel aparte. En este caso, la tira que contiene el valor es copiada pero *no destruida*. **Definición 6** * *

(iv) Liberar un sitio de almacenamiento; es decir, quitar la marca y destruirle el valor asociado. **Definición 7** * *

Nos detendremos ahora, para examinar la forma en que este modelo para el elemento de almacenamiento de un sistema de computación está reflejado en los servicios disponibles al Servicio de Computación para el estudiante.

8.3.2 Introducción al Laboratorio de programación

8.3.2 Introducción * *

Usted conoce seguramente la idea subyacente a un *laboratorio de idiomas* el cual utiliza la técnica moderna para suministrar un conocimiento práctico de un idioma en el menor tiempo posible. Nuestro objetivo en

el Laboratorio de programación es parecido, con la diferencia de que buscamos enseñar un lenguaje que se pueda utilizar para suministrar un algoritmo a un computador. Por esta razón utilizamos lo que se llama un *lenguaje de programación.*

Un computador es un equipo electrónico altamente sofisticado; será suficiente, sin embargo, basar nuestras consideraciones en el modelo conceptual sencillo de computador que estamos desarrollando en esta unidad. Como todos los modelos conceptuales, éste es una abstracción; contiene únicamente aquellas características estructurales que necesitamos tener en cuenta con el fin de entender el proceso de construcción de un algoritmo, en tal forma que éste pueda ser ejecutado por un computador. En el programa de televisión asociado a esta unidad, desarrollaremos un modelo conceptual alternativo cuya meta es ayudarlo a comprender la estructura y operación del computador mismo.

Los textos del Laboratorio de programación forman la base de su trabajo práctico de computación. Usted debe trabajar cuidadosamente cada parte del texto de laboratorio con la ayuda de un terminal de computación (o enviando por correo el material que debe ser procesado, al Servicio de Computación para el estudiante). Los detalles prácticos al uso de los terminales del Servicio de Computación para el estudiante y el servicio de correos se pueden encontrar en el documento *Servicio de Computación para el estudiante - Guía del usuario* que se le ha suministrado.

Cada parte del texto del Laboratorio de programación se supone que es equivalente a *una semana de prácticas de computación* las cuales deben ocuparle aproximadamente una hora de su tiempo (casi 15 minutos de esa hora se deben emplear en el terminal de computación). Usted puede trabajar en estas secciones tanto como quiera, aunque debiera aspirar a completar las cinco partes incluidas en esta unidad durante las próximas cinco semanas. Probablemente encontrará muy conveniente leer la *totalidad* de esta unidad *antes* de enfrentarse a la primera sesión práctica.

Una sucesión de instrucciones, ejecutables por un computador, se llama un **programa** . (El deletreo es deliberado: "program" es un término técnico y es diferente de la palabra inglesa común, "programme".) **Definición 1** * * *

La tarea de planear y escribir una serie de instrucciones para ejecutar un trabajo (es decir, para que corresponda a un algoritmo particular) se llama **programación** . El lenguaje en el cual están escritas las instrucciones se llama un **lenguaje de programación** . **Definición 2** * * **Definición 3** * *

Le enseñaremos a escribir programas en un lenguaje de programación sencillo, llamado **BASIC** *. (Las letras significan **Beginner's All-purpose Symbolic Instruction Code** que en español significan Código de instrucción simbólica para utilidad general de los principiantes) —pero el nombre en sí mismo es más indicativo de la naturaleza del lenguaje). Las ventajas de este lenguaje consisten en que es fácil de aprender y es disponible para su uso en una amplia gama de sistemas de computación existentes. Uno de tales sistemas de computación forma la base del Servicio de Computación para el estudiante.

* El BASIC fue desarrollado en la Universidad de Dartmouth, New Hampshire, U.S.A. por el profesor J. G. Kemeny y el profesor T. E. Kurtz.

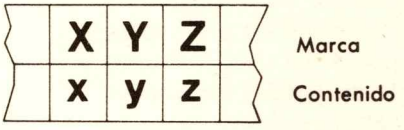
8.3.3 Laboratorio de programación: Parte 1, constantes, variables y asignaciones

8.3.3

Tema principal
* * *

En la Sección 8.3.1 vimos que un computador contiene un elemento de almacenamiento que es utilizado para almacenar la información que se manipula. En nuestro modelo, el elemento de almacenamiento del computador es una colección de sitios de almacenamiento del computador, cada uno de los cuales es capaz de contener una parte de la información. Lo que constituye una parte de la información depende de la clase de información particularmente involucrada, pero de todos modos cada sitio de almacenamiento puede contener un número decimal.

Considérense tres sitios de almacenamiento dentro del computador, los cuales han sido marcados como X, Y y Z (esto es, han sido declaradas unas variables llamadas x, y, y z) tal como se muestra en el siguiente diagrama:



Hemos escrito la letra mayúscula, X, sobre el sitio de almacenamiento, con el fin de dar un nombre a la localización misma, y hemos escrito la letra minúscula x, dentro del sitio mismo, con el fin de dar un nombre al contenido de la localización. Puede ser que usted no piense que sea particularmente importante el ser capaz de referirse por separado a la localización y a su contenido, pero esta distinción es de importancia fundamental y nos ayudará en la exposición siguiente. Los símbolos X, Y, Z son nombres escogidos para identificar los propios sitios de almacenamiento y los símbolos x, y, z son los valores actuales de las variables almacenados en dichos sitios. En el lenguaje de la computación, los nombres X, Y, Z se llaman las direcciones de las localizaciones.

Definición 1
* *

En el lenguaje de programación BASIC, los nombres de los sitios de almacenamiento están compuestos o bien de una letra mayúscula o bien de una letra mayúscula seguida de un número dígito; así, A7, B, M1, y W son ejemplos de nombres. Un sitio de almacenamiento se asigna a cada nombre empleado y todos los cálculos comprenden los valores actuales *contenidos* en tales sitios. Las letras minúsculas no figuran en la mayor parte de los dispositivos de entrada y, en consecuencia, en lenguaje BASIC debemos referirnos siempre a las variables por medio de letras mayúsculas que son las que representan los nombres de las respectivas localizaciones. Daremos una ilustración de cómo trabaja esto.

En el lenguaje BASIC, los valores numéricos se pueden representar de dos maneras: o bien en forma de una constante, o bien como el contenido de un sitio de almacenamiento especificado (es decir, como el valor actual de una variable con nombre). Las constantes en un programa, se escriben en la forma usual para los números decimales, con la excepción de que las comas, utilizadas algunas veces para separar grupos de tres dígitos, se eliminan (por ejemplo, 5,791 se escribe 5791 en BASIC). Una variable puede tener asociada a ella un solo valor si una parte de la información ha sido asignada al sitio de almacenamiento asociado a dicha variable. En el lenguaje de programación BASIC tenemos una notación

para especificar la acción de asignar un valor a una variable. La frase de asignación

x se convierte en 2

la cual corresponde a asignar el valor 2 a la variable *x* en el sitio X, se escribe en BASIC como

```
LET X = 2
```

Esta orden puede ser interpretada como

Hágase que el contenido del sitio X se haga igual al valor numérico 2

o, más sencillamente, como

Hágase que el valor de X se convierta en 2

En función de nuestro modelo, el valor numérico 2 es escrito en una tira de papel y almacenado en el sitio marcado X.

Obsérvese que esta frase es un *imperativo* (esto es, una orden para efectuar la instrucción enunciada). Es importante que quede en claro que esta frase no significa lo mismo que la *ecuación* $x = 2$, ya que esta última es un enunciado indicativo que se debe interpretar como

la variable *x* tiene el valor 2

o, más sencillamente, como

x es 2

En consecuencia

$x = 2$

es una afirmación acerca del valor existente de *x*, mientras que

```
LET X = 2
```

es una asignación de un valor a la variable *x*.

Considere la ejecución de la instrucción

```
LET X = 5
```

seguida a continuación por la instrucción

```
LET X = 9
```

En este punto un nuevo valor, 9, se asigna a la variable llamada *x* y el valor antiguo, 5, de esta variable queda destruido, ya que cada sitio de almacenamiento puede contener sólo un valor (ítem de información). Por tanto, cada nueva asignación de un valor a una variable *destruye* el valor existente de la variable y lo remplaza por el nuevo valor.

Cuando definíamos el elemento de almacenamiento en la Sección 8.3.1, dijimos también que una variable podía ser copiada; esta instrucción se especifica en la programación BASIC escribiendo el nombre de la variable al lado derecho del símbolo = como sigue:

La instrucción

```
LET X = Y
```

se interpreta como

Hágase que el contenido del sitio X se iguale al contenido del sitio Y

o simplemente

Hágase que el valor de X se convierta en una copia del valor Y

Sean las instrucciones

```
LET Y = Z
LET X = Y
```

Después de que ambas instrucciones se han ejecutado, *x*, *y*, y *z* tendrán todas el mismo valor; en este caso igual al valor original de la variable *z*.

Supongamos, sin embargo, que ejecutamos las instrucciones

```
LET X = Y
```

y

```
LET Y = Z
```

Entonces *x* toma el valor original de *y*, y *y* toma el valor original de *z*, y dichos valores pueden ser diferentes. Vemos entonces que el *orden* en el cual son asignados los valores es de vital importancia; por tanto, usted debe ser muy cuidadoso para especificar las instrucciones en el orden correcto.

Ejercicio práctico 1

Ejercicio práctico 1

Las características operativas del Servicio de Computación para el estudiante se deben estudiar detalladamente (leyendo la *Guía del usuario*) *antes* de comenzar el trabajo práctico.

Debe trabajar cuidadosamente este ejercicio usando lápiz y papel y a continuación procesar sus respuestas en un terminal de computación utilizando el programa DEMON dando como entrada:

```
GET — 8DEMON
RUN
```

DEMON averiguará por el número del ejercicio, que en este caso es 1, y a continuación pedirá su respuesta. Si está utilizando el correo debe presentar sus respuestas escritas con claridad en una hoja de codificación en la forma especificada en la *Guía del usuario* (indique simplemente el ejercicio práctico y los números importantes del problema con sus respuestas) y diríjalo a la dirección dada en la *Guía del usuario*. Si se presentan problemas relacionados con la parte operacional del trabajo práctico, debe consultar la sección correspondiente de la *Guía del usuario*.

1 Escriba los siguientes números como constantes en BASIC:

(a) 1 726 (b) $\frac{4}{5}$ (c) $\frac{2}{7}$,

si dichas constantes se deben almacenar correctamente con seis cifras significativas

2 Sólo una de las siguientes marcas es válida en BASIC. Indicar si es (A), (B) ó (C).

(A) A3 (B) 2D (C) PI

3 ¿Cuál es el valor de la variable *s* después de la ejecución de la siguiente sucesión de instrucciones?

```
LET A = 4
LET B = 3
LET A = B
LET S = B
```

4 Escriba una sucesión de *tres* instrucciones de asignación en BASIC para intercambiar los valores que están ahora en los sitios A y B. ¿Puede obtener el mismo resultado con menos de tres instrucciones?

■

8.3.4 El concepto de ejecución de un trabajo: Elemento de procesamiento

8.3.4

Tema principal
* *

El elemento de procesamiento de nuestro modelo consta de un conjunto de facilidades que nos permiten ejecutar instrucciones individuales y servirnos de guía a través de un algoritmo. El elemento de procesamiento puede ser concebido como un grupo formado por tres esclavos infrahumanos (robots) cada uno de los cuales lleva a cabo un conjunto limitado de tareas. Sus oficios generales se pueden describir como sigue:

(i) El mensajero: El oficio de este esclavo consiste en llevar información de un sitio a otro. (Podemos imaginárnoslo equipado con una libreta de apuntes en la que anota la información trasferida.)

(ii) El calculador: Las funciones de este esclavo consisten en ejecutar instrucciones elementales utilizando la información suministrada por el mensajero con el fin de producir nuevas informaciones.

(iii) El supervisor: La tarea de este esclavo consiste en interpretar las instrucciones que especifican al algoritmo (podemos considerar por ahora que esta especificación es un diagrama de flujos) y asignar tareas específicas a todos los esclavos.

Es importante recordar que estamos creando un modelo conceptual de un sistema de computación y, por tanto, los esclavos con los cuales estamos construyendo el elemento de procesamiento no deben tener atributos de carácter humano; en particular, no tienen memoria (la memoria es suministrada por el elemento de almacenamiento conceptual descrito anteriormente) y tampoco tienen inteligencia o libre albedrío. Ellos ejecutan una gama limitada de instrucciones con una obediencia total.

En este estadio de la creación de nuestro modelo no deseamos especificar en detalle el conjunto total de instrucciones que cada uno de los esclavos puede ejecutar ya que esta descripción puede variarse con el fin de crear modelos de máquinas capaces de ejecutar instrucciones en grados variables de dificultad. En cambio examinaremos la ejecución de una o dos tareas sencillas y veremos qué variedad de funciones requieren éstas de los esclavos.

Consideremos la ejecución de la sencilla instrucción de asignación que encontramos en la sección anterior. Comencemos a partir de una situación en la cual todos los sitios de almacenamiento están desocupados (es decir, sin marca) y consideremos la ejecución de la siguiente sucesión de instrucciones

```
1 LET X = 3
2 LET Y = X
```

donde los enteros 1 y 2 son números de fila que pueden servir para identificar la instrucción que se está ejecutando. La ejecución de esas instrucciones se puede seguir dibujando en una tabla parecida a la utilizada anteriormente; en este caso, sin embargo, estamos interesados en poner en la tabla la información concerniente a la operación de nuestro sistema modelo. Una tabla conveniente puede ser ordenada como sigue:

Etapa	Acción del supervisor	Acción del mensajero
Etapa 1	Identifica una instrucción de asignación; comienza a calcular el valor que debe ser asignado.	
	Copia el valor en una tira de papel. Se asegura de que el valor que debe ser asignado ha sido completamente evaluado.	
	Da la tira de papel con el valor escrito, al mensajero con la instrucción de almacenarlo en un sitio llamado X.	
		Va a la unidad de almacenamiento; encuentra que no hay ningún sitio marcado X; crea una marca y marca un sitio con X; deposita la tira con el valor en el sitio X y vuelve al supervisor.
Etapa 2	Identifica una instrucción de asignación; comienza el cálculo del valor que debe ser asignado.	
	Envía al mensajero con la instrucción de copiar dicho valor en el sitio X.	
		Va a la unidad de almacenamiento; escoge el sitio X; copia el valor almacenado en X en una tira de papel nueva y vuelve con este valor donde el supervisor.
	Reconoce que el valor que debe ser asignado ha sido ahora completamente evaluado.	
	Devuelve la tira al mensajero con la instrucción de depositarla en el sitio Y.	
		Va a la unidad de almacenamiento, encuentra que no hay ningún sitio marcado Y; crea una marca y marca un sitio con Y; almacena la tira en Y y vuelve donde el supervisor.

La sucesión se puede continuar si hay más instrucciones que deban ser ejecutadas. Observe que en el proceso de ejecución de una tarea el supervisor divide cada instrucción en un conjunto cuidadosamente coordinado de instrucciones más sencillas. Dichas instrucciones primitivas se ejecutan secuencialmente (es decir, la una después de la otra en forma cuidadosamente coordinada) y de acuerdo con esto el estado interno de nuestro modelo cambia en forma ordenada. Esta forma ordenada de operar es característica de los computadores digitales modernos, aunque en el caso de un sistema de computación real la actividad ejecutada por una sola instrucción puede ser mucho más pequeña y, en consecuencia, la trasferencia de mensajes es mucho más numerosa. Este punto será tratado en forma más destacada por medio del modelo de computador que examinaremos en el programa de TV asociado a esta unidad. Usted encontrará que la técnica descrita anteriormente le será muy útil para ayudarlo a entender el efecto de instrucciones particulares.

En la descripción anterior el esclavo-calculador no fue utilizado durante la ejecución de la sucesión de instrucciones. En consecuencia, debemos considerar el papel que desempeña este componente del procesador de datos. En matemáticas una *expresión* es algo de la forma $x^2 - 4yz$; es decir, una sucesión o cadena correctamente formada de variables, constantes y símbolos especiales utilizados para indicar operaciones binarias. Si nuestro sistema debe ser capaz de ejecutar operaciones que comprendan tales expresiones, entonces el calculador debe ser capaz de ejecutar las operaciones binarias básicas de la aritmética sobre pares de valores suministrados por el supervisor.

Consideremos, por ejemplo, la evaluación de la expresión X + Y, la cual es interpretada como $x + y$, es decir, sumar el contenido que hay en el sitio X al contenido del sitio Y. Empleando la notación introducida anteriormente, la ejecución de esta etapa puede interpretarse como sigue:

Etapa	Supervisor	Mensajero	Calculador
Evaluar X + Y	Identifica una expresión aritmética. Envía al mensajero a copiar el valor que hay en X.		
		Va a la unidad de almacenamiento; localiza el sitio X; copia el valor x en una tira de papel y se devuelve.	
	Recibe la tira y copia el valor en una hoja de instrucciones; envía al mensajero a copiar el valor que hay en Y.		

Etapa	Supervisor	Mensajero	Calculador
		Va a la unidad de almacenamiento; localiza el sitio Y; copia el valor *y* en una tira de papel y se devuelve.	
	Recibe la tira de papel; copia el valor en una hoja de instrucciones y especifica la operación "sume". Da al mensajero la hoja de instrucciones; envía al mensajero a donde el calculador diciéndole que espere por la respuesta.		
		Va donde el calculador; le entrega la hoja de instrucciones; espera por la respuesta.	
			Ejecuta la operación especificada con los valores recibidos; copia el resultado en una tira de papel y se la entrega al mensajero.
		Va donde el supervisor con la tira de papel.	
	Observa que la evaluación se ha completado.		

8.3.5 Laboratorio de programación: Parte 2, operaciones aritméticas sencillas

8.3.5

Tema principal
* * *

En la primera parte del texto del Laboratorio de programación expusimos los significados de constantes, variables y asignaciones en el lenguaje de programación BASIC en relación con la unidad de almacenamiento de nuestro modelo de sistema de computación. Encontramos

allí nuestra primera instrucción en el lenguaje de programación BASIC: una asignación sencilla.

En esta parte del texto del Laboratorio de programación, deseamos desarrollar esta sencilla instrucción de asignación para incorporar la operación del componente calculador del elemento de procesamiento; en particular, consideramos las operaciones aritméticas sencillas ejecutadas por el componente de cálculo.

Como antes, consideramos tres sitios dentro del elemento de almacenamiento, llamados X, Y y Z. En BASIC podemos definir cinco operaciones aritméticas ejecutables entre pares de sitios de almacenamiento, como se muestra en la siguiente tabla:

Operación binaria	Notación BASIC	Significado
Suma n	X + Y	Sume el contenido del sitio X al contenido del sitio Y, es decir, encuentre $x + y$.
Resta	X − Y	Encuentre $x - y$.
Producto	X * Y	Encuentre $x \times y$.
Cociente	X/Y	Encuentre x/y.
Exponenciación	X ↑ Y	Encuentre el valor x^y, esto es, el contenido del sitio X elevado a la potencia especificada por el contenido del sitio Y.

Podemos ahora dar un significado a una declaración de la forma:

LET Z = X + Y

por ejemplo,

Hágase que el valor en Z sea el valor obtenido al sumar el contenido del sitio X al valor del contenido en el sitio Y

o alternativamente,

Sume el valor de *x* al valor de *y* y coloque el resultado en el sitio Z

En la primera parte del texto del Laboratorio de programación descubrimos que las constantes pueden formar la parte numérica de una instrucción de asignación sencilla; en forma parecida, una constante puede remplazar a uno o ambos sitios de almacenamiento en una operación aritmética sencilla. Entonces, en la tabla dada anteriormente, o X, o Y, o ambos X y Y, se pueden remplazar por números decimales o números enteros.

Como todos los cómputos pueden, en última instancia, ser remplazados por una sucesión de operaciones aritméticas simples, tales como las que hemos descrito, estamos ahora en condiciones de escribir una sucesión de instrucciones para especificar la ejecución de un cómputo numérico arbitrariamente complejo. Antes de intentar resolver algunos ejercicios, consideremos un ejemplo. ¿Qué significado puede dársele a la instrucción

LET X = X + 1?

Comenzamos por evaluar la expresión que está a la derecha del signo igual. Su valor se obtiene copiando el valor contenido en el sitio X y

agregando 1 a este valor. El resultado de este cálculo se convierte en un nuevo valor de x (esto es, es asignado al sitio X). En consecuencia, esta instrucción debe entenderse como hacer que el sitio X se aumente en 1.

Resumen

Resumen
* * *

Las operaciones que pueden ser ejecutadas por nuestro modelo de elemento de procesamiento de un sistema de computación están reflejadas por las características del lenguaje de programación BASIC y viceversa. Ciertas características del BASIC se pueden interpretar en función de nuestro modelo de elemento de procesamiento como sigue:

Ejecución secuencial: Cada instrucción se subdivide en un conjunto de instrucciones más simples que se pueden ejecutar por los componentes constituyentes del procesador. La instrucción en BASIC debe especificar exactamente las acciones que se deben ejecutar.

Operaciones aritméticas: Un conjunto de operaciones aritméticas se pueden aplicar a un par de valores, y el valor resultante se puede asignar a un sitio de almacenamiento. Esto nos conduce hacia una definición ampliada de la instrucción de asignación en BASIC, incorporando una expresión aritmética sencilla como el valor que debe asignarse.

Ejercicio práctico 2

Ejercicio práctico 2

Debe trabajar cuidadosamente en este ejercicio usando lápiz y papel y a continuación debe procesar sus respuestas en un terminal de computación por medio del programa de biblioteca DEMON poniendo:

```
GET — 8DEMON
RUN
```

En este caso el número del ejercicio es 2. Si está utilizando el servicio de correos debe presentar sus respuestas claramente escritas en una hoja de codificación en la forma especificada en la *Guía del usuario.*

1 Escriba las siguientes expresiones en notación BASIC (sin paréntesis):

(a) 2^b (b) xy (c) $\frac{1}{3}w$

2 Calcular las siguientes expresiones en BASIC, donde $x = 13$ y $n = 3$ (dando los resultados con seis cifras significativas)

(a) X * 5 (b) X ↑ N (c) X/N

3 Suponiendo inicialmente que los valores que hay en A, B, C, son 2, 3, 4, respectivamente, calcule el valor de s y t "siguiendo la ejecución" de la siguiente sucesión de instrucciones en BASIC:

```
LET S = A + C
LET T = A + B
LET S = T/S
LET T = S ↑ 2
```

4 Escriba una sucesión de instrucciones en BASIC para calcular una cantidad x dada por la ecuación $\frac{1}{x} = \frac{1}{y^n} - \frac{1}{z^n}$ si los valores de y, z y n son conocidos. ■

8.3.6

Tema principal
* *

8.3.6 Evaluación de expresiones

Como vimos en la sección anterior, el elemento calculador de nuestro modelo de procesador es capaz de combinar pares de valores para ejecutar una gama de operaciones aritméticas sencillas y esta facilidad nos permite definir un programa para evaluar una expresión aritmética arbitrariamente compleja. Sin embargo, si la programación de nuestro sistema debe ser razonablemente fácil y conveniente, necesitamos de una comodidad más sofisticada que nos permita presentar expresiones aritméticas arbitrariamente complejas en la forma de *una sola* instrucción.

La razón por la cual es necesaria tal comodidad puede deducirse a partir del siguiente argumento. La sucesión de instrucciones:

```
LET S = 1 - X
LET T = 1 + X
LET S = S/T
LET E = S↑N
LET S = 2 - Z
LET T = 2 + Z
LET S = S / T
LET S = S ↑ M
LET E = E*S
```

escrita en la notación que se introdujo en la sección anterior, corresponde a la evaluación de e por la fórmula

$$e = \left(\frac{1-x}{1+x}\right)^n \times \left(\frac{2-z}{2+z}\right)^m$$

y a colocar el resultado en el sitio E. La ejecución de este cálculo con las nueve instrucciones simples enunciadas antes tiene las siguientes desventajas:

(i) Se emplea bastante trabajo de escritorio y el usuario tiene muchas oportunidades de cometer errores.
(ii) Al subdividir la expresión en sus componentes primitivas se destruye su estructura interna y así se hace la verificación de la especificación mucho más difícil.

El uso de una sola instrucción, más compleja, resuelve estos problemas, pero exige que el usuario aprenda cómo especificar las expresiones en tal forma que el elemento de procesamiento pueda interpretarlas correctamente. Como veremos, la instrucción única correctamente especificada es

```
LET E = (((1 - X)/(1 + X))↑N)*(((2 - Z)/(2 + Z))↑M)
```

Observe que cuando cambiamos de la notación matemática a nuestra forma ampliada de notación de programación, se introducen seis pares extras de paréntesis; si cualesquiera de estos paréntesis se omiten entonces la evaluación no será ejecutada correctamente. Con el fin de entender cómo debe especificarse una expresión compleja, debemos tener en cuenta las reglas por medio de las cuales nuestro modelo de elemento de procesamiento evalúa tales expresiones.

Consideremos primero el cálculo de expresiones que no contienen paréntesis. En este caso el *orden de evaluación* está determinado por un conjunto de *reglas de precedencia.* Usted aprendió dichas reglas cuando se le enseñó la aritmética elemental; el efecto de las reglas de precedencia es determinar el orden en el cual las operaciones aritméticas se llevan a cabo. Por ejemplo, usted evaluaría la expresión

$$4 + 3 \times 2$$

formando el producto y sumando a 4 el resultado para obtener el valor 10, en lugar de hacer la suma y multiplicar por 2 el resultado para obtener 14. Entonces será consistente con sus nociones intuitivas actuales si el elemento de procesamiento interpreta la expresión

A + B * C

como formado por el producto de los valores de las variables *b* y *c* y agregando el resultado al valor de la variable *a*.

Una tabla completa de reglas de precedencia para evaluar expresiones aritméticas se da a continuación.

Reglas de precedencia
* * *

Operación	Notación BASIC	Significado	Precedencia
Suma	X + Y	$x + y$	Primera
Resta	X − Y	$x - y$	
Producto	X * Y	$x \times y$	Segunda
Cociente	X/Y	x/y	
Tomar el negativo	−X	$-x$	
Exponenciación	X ↑ Y	x^y	Tercera

La variable *x* o la variable *y* o ambas, se pueden remplazar por constantes decimales. En esta tabla introdujimos una nueva operación (es decir, una operación no incluida en la tabla dada en la Sección 8.3.5) y esta es la operación unitaria llamada *tomar el negativo.* Esta operación puede interpretarse como si se supusiera que el valor numérico contenido en un sitio de almacenamiento consistiera de un signo (+, ó, −) y un valor numérico absoluto; en consecuencia, la operación de tomar el negativo consiste en cambiar el signo en el sitio de almacenamiento apropiado. Por ejemplo, exigimos que la instrucción

LET Y = −X

copie el valor que hay en el sitio X, cambie el signo (de la copia), y almacene el resultado como el valor del sitio Y. (Observe que el valor que hay en el sitio X *no* cambia de signo.)

Utilizando la tabla de precedencias podemos definir una regla sencilla para evaluar una expresión aritmética que no contiene paréntesis, como sigue:

Verifique la expresión de *izquierda a derecha* para ver si hay operaciones del *primer nivel.* Cuando encuentre tal operación, ejecute la evaluación que ésta especifique y comience a verificar de nuevo a partir del punto en el que se interrumpió. Cuando se hayan efectuado todas las operaciones del primer nivel (es decir, cuando la verificación se ha completado), repita de izquierda a derecha la comprobación para operaciones del *segundo nivel* y así sucesivamente.

Pueden especificarse reglas más sofisticadas para evitar esta verificación repetida; sin embargo, la regla anterior nos capacita para construir, de manera sencilla, el proceso de evaluación. Considere el siguiente ejemplo †:

LET X = −A + B − C * D/E ↑ F

donde *a*, *b*, *c*, *d*, *e*, y *f* tienen los valores 6, 6, 5, 4, 2, y 2 respectivamente. La evaluación de la expresión puede ser seguida como a continuación se muestra:

Etapa	Acción	Resultado	Comentario
1	Comienza verificación de izquierda a derecha para operaciones del primer nivel.		
2	Calcula E ↑ F.	LET X = −A + B − C * D/4	Completada primera verificación.
3	Comienza verificación de izquierda a derecha para operaciones del segundo nivel.		
4	Calcula − A.	LET X = (−6) + B − C * D/4	Sigue la verificación.
5	Calcula C * D.	LET X = (−6) + B − 20/4	Sigue la verificación.
6	Calcula 20/4.	LET X = (−6) + B − 5	Completada la segunda verificación.
7	Comienza verificación de izquierda a derecha para operaciones del tercer nivel.		
8	Calcula (− 6) + B.	LET X = 0 - 5	Sigue la verificación.
9	Calcula 0 − 5.	LET X = (−5)	Completada la tercera verificación.
10	Dé a X el valor resultante.		

Nota

Los paréntesis se ponen alrededor de las constantes negativas para destacar el hecho de que cada valor consiste de un signo y de un valor absoluto en tal forma que el signo menos en −6, por ejemplo, es parte del valor de la constante.

La tabla puede interpretarse en función de las operaciones ejecutadas por los elementos del procesador introducidos en la sección anterior, ya que cada cálculo comprende una expresión aritmética sencilla que, a su vez, incorpora una única operación aritmética.

Como se habrá dado cuenta, los paréntesis se introducen dentro de una expresión para indicar el orden de la evaluación. Por ejemplo, usted podría calcular automáticamente

(4 + 3) × 2

† En este ejemplo, el primer signo menos es una operación *unitaria* y el segundo signo menos es una operación *binaria*. Si el menos unitario es necesario en la mitad de una expresión, *debe* ser precedido a la izquierda por un paréntesis.

para obtener el resultado 14 (sumando antes de formar el producto). La regla para evaluar expresiones que contengan paréntesis puede describirse como sigue:

1 Ponga un contador en cero y verifique la expresión izquierda a derecha en búsqueda de paréntesis. Cuando encuentre un paréntesis izquierdo, aumente el contador en una unidad y marque el paréntesis con el valor que registre el contador; cuando encuentre un paréntesis derecho, márquelo con el valor actual del contador, disminuya la cuenta en una unidad y continúe verificando. (La cuenta debe ser igual a cero cuando se acaba la verificación.) Escriba el valor máximo que ha alcanzado el conteo.

2 Una *subexpresión* es una sucesión de símbolos encerrados entre paréntesis izquierdos y derechos adyacentes que tengan valores iguales. Calcule la expresión aplicando la regla de evaluación de expresiones sin paréntesis a sus subexpresiones, comenzando con aquellas que estén encerradas por paréntesis que tengan máximo valor, por medio de una sucesión de verificaciones de izquierda a derecha.

Definición 1
*

Para ver cómo opera esta regla, consideremos el ejemplo anterior:

$$\text{LET E} = (((1 - X)/(1 + X)) \uparrow N) * (((2 - Z)/(2 + Z)) \uparrow M)$$

con $x = 2$, $n = 4$, $z = 3$, y $m = 5$.
La aplicación de la primera regla nos da:

$$\text{LET E} = (\underset{123}{(}\underset{3}{(}1 - X\underset{3}{)}/\underset{3}{(}1 + X\underset{32}{)}\underset{32}{)} \uparrow N\underset{1}{)} * (\underset{123}{(}\underset{3}{(}2 - Z\underset{3}{)}/\underset{3}{(}2 + Z\underset{32}{)}\underset{32}{)} \uparrow M\underset{1}{)}$$

y un valor máximo para el conteo es 3. La aplicación de la segunda regla se efectúa como sigue:

Etapa	Acción	Resultado	Comentario
1	Se comienza verificación para subexpresiones de orden 3.		
2	Calcula $(\underset{3}{1} - \underset{3}{X})$ por la regla de expresiones sin paréntesis.		
3	Calcula $1 - X$.	$\text{LET E} = (\underset{12}{(}\underset{3}{(}-1)/\underset{3}{(}1 + X\underset{32}{)}\underset{32}{)} \uparrow \cdots$	Sigue verificando.
4	Calcula $(\underset{3}{1} + \underset{3}{X})$ por la regla de expresiones sin paréntesis.		
5	Calcula $1 + X$.	$\text{LET E} = (\underset{12}{(}\underset{2}{(}-1)/3\underset{2}{)} \uparrow \cdots$	Sigue verificando.
6–9	Calcula subexpresiones de orden 3 restantes.	$\text{LET E} = (\underset{12}{(}\underset{2}{(}-1)/3\underset{2}{)} \uparrow N\underset{1}{)} * (\underset{12}{(}\underset{2}{(}-1)/5\underset{2}{)} \uparrow M\underset{1}{)}$	Fin de la verificación.
10	Comienza verificación para subexpresiones de orden 2 y así sucesivamente.		

8.3.7 Laboratorio de programación: Parte 3, expresiones aritméticas en lenguaje BASIC

8.3.7

Tema principal
* * *

El lenguaje de programación BASIC permite al usuario escribir instrucciones de asignación en las cuales la parte que se va a evaluar es una expresión aritmética formada por constantes, nombres de localizaciones y operaciones aritméticas. Estas se evalúan de acuerdo con las reglas descritas en la sección precedente. Una especificación completa de esta forma general de instrucción de asignación está dada en las secciones correspondientes de la *Guía del usuario.* Como ejemplo, considere la instrucción de asignación:

```
LET  X = C - (A + D) / (B ↑ E) * F
             1       2
             3
                     4
             5
```

Los corchetes horizontales muestran el orden en el que se calculan las subexpresiones por medio de las reglas de evaluación definidas en la sección anterior. Esta única instrucción en BASIC es equivalente a las siguientes cinco instrucciones simples en BASIC del tipo expuesto en la Parte 2 del texto de Laboratorio de programación:

```
LET P = A + D
LET Q = B↑E
LET R = P/Q
LET S = R * F
LET X = C - S
```

donde P, Q, R, S son sitios utilizados temporalmente como almacenamiento de resultados parciales.

Si tiene usted alguna duda acerca del efecto exacto de las reglas de cálculo de una expresión, introduzca entonces un suficiente número adicional de paréntesis para definir unívocamente el proceso de evaluación requerido.

El valor de una variable numérica en el sistema de programación BASIC disponible al Servicio de computación para el estudiante es o bien un número decimal finito comprendido en el rango aproximado de 10^{-38} a 10^{+38}, o cero. (Realmente el máximo es $1{,}70141 \times 10^{38}$.) La precisión de cada número es de seis cifras decimales, y cada número tiene signo (es decir, el valor absoluto de la variable está almacenado junto con un indicador que define su signo). En consecuencia el conjunto de valores numéricos se extrae del conjunto de números de seis dígitos definido por

$$\{x : -10^{38} \leqslant x \leqslant -10^{-38} \text{ o } x = 0 \text{ o } 10^{-38} \leqslant x \leqslant 10^{38}\}$$

Con el fin de ser capaces de expresar números muy grandes o muy pequeños de este conjunto usando sólo seis dígitos decimales, necesitamos adoptar algunas convenciones. Por ejemplo, el número 86.400.000 no puede ser dado como dato de entrada directamente ya que contiene más de seis dígitos. Sin embargo, este número puede escribirse como $8{,}64 \times 10^{7}$, donde 8,64 se llama la *mantisa*, 10 es la *base* y 7 es el *expo-*

Definición 1
*

nente. Este valor puede ser representado en BASIC por 8,64E7. La representación se llama formato normalizado de punto flotante. La normalización consiste en escribir la mantisa como una constante decimal en el intervalo [1, 10[. La normalización tiene la ventaja de hacer que la notación sea única y también de máxima precisión ya que pueden utilizarse hasta seis dígitos para representar la mantisa.

Definición 2
* *

La mantisa y el exponente pueden conservarse en el mismo sitio de almacenamiento; el primero es un número decimal con signo compuesto por seis dígitos en el intervalo [1, 10[, y el último es un entero con signo comprendido entre 0 y 38.

Resumen

Resumen
* *

La evaluación de una expresión aritmética compleja puede ser reducida sistemáticamente en una sucesión de operaciones aritméticas sencillas ejecutables por nuestro modelo de elemento de procesamiento de un sistema de computación. Esta facilidad nos capacita para crear e interpretar expresiones aritméticas complejas como parte numérica de una instrucción de asignación en BASIC.

La notación utilizada para describir constantes numéricas en BASIC ha sido ampliada con la introducción de una notación para manejar el signo de la constante y un formato normalizado de punto flotante para permitir la descripción de valores muy grandes y muy pequeños.

Ejercicio práctico 3

Ejercicio práctico 3

Debe trabajar cuidadosamente este ejercicio con lápiz y papel y, a continuación, procesar sus respuestas en un terminal de computación por medio del programa almacenado DEMON, poniendo:

```
GET —8DEMON
RUN
```

En el caso presente el número del ejercicio es 3. Si usa el servicio de correo debe presentar sus respuestas claramente escritas en una hoja de codificación en la forma especificada en la *Guía del usuario*.

1 Escriba las siguientes constantes matemáticas en el formato normalizado de punto flotante:

(a) −0,000 031 42
(b) 14770200
(c) $3{,}0091 \times 10^{10}$

2 Calcule el valor de *s*, en formato normalizado de punto flotante, que se encuentra ejecutando la siguiente sucesión de instrucciones:

```
LET A = 5,607E − 1
LET S = 8,372E − 9 + 3,142 * A
```

3 Escriba una o más instrucciones en BASIC para calcular el valor *y* a partir de la fórmula

$$y = a + bx + cx^2 + dx^3$$

4 Aplique las reglas de evaluación de expresiones aritméticas descritas en la Sección 8.3.6 para reducir la declaración:

```
LET X = −A + B/C + (D − E * F)↑2
```

a una sucesión apropiada de instrucciones de asignación sencillas de las cuales cada una comprende una sola operación aritmética. ■

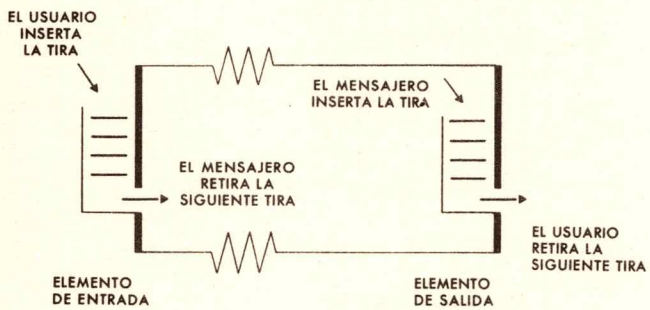
8.3.8 Comunicación con el usuario

8.3.8

Tema principal
* *

Hasta ahora, nuestro modelo de sistema de computación ha constado de un elemento de almacenamiento y un elemento de procesamiento. Sin embargo, las operaciones llevadas a cabo dentro de estos elementos tienen lugar en el interior del sistema de computación. Usted sabe por propia experiencia con el Servicio de Computación para el estudiante, que es posible dar información a un computador y recibir información del mismo o bien directamente a través del terminal, o bien indirectamente por medio de las hojas de codificación y las hojas de resultados (la salida devuelta al estudiante; vea la *Guía del usuario* para la descripción apropiada). En consecuencia, debemos agregar elementos de entrada y de salida a nuestro modelo para que pueda llevar a cabo las funciones de comunicación.

Imaginémonos que los elementos de almacenamiento y procesamiento de nuestro modelo de computador están totalmente encerrados en un cuarto sellado y que la comunicación con el ambiente exterior tiene lugar a través de dos ventanillas construidas tal como se muestran en el diagrama que sigue:



Las operaciones básicas ejecutadas por estos elementos son:

Entrada: 1 El usuario puede apilar las tiras listas para entrar, poniéndolas en el elemento de entrada. Cada tira contiene un ítem de información la cual corresponde a una línea de entrada en el terminal o en una hoja de codificación.

2 El supervisor puede enviar al mensajero a recoger la *siguiente tira* del elemento de entrada.

Salida: 1 El supervisor puede enviar al mensajero a apilar las tiras listas para salir poniéndolas en el elemento de salida. Cada tira contiene un ítem de información que corresponde a una línea de salida en el terminal o en una hoja de resultados.

2 El usuario puede sacar la *siguiente tira* del elemento de salida.

Es importante observar que estas operaciones *conservan el orden*; es decir, el sistema acepta tiras de entrada exactamente en el orden en el cual el usuario las suministra y viceversa para la salida. Como se dará cuenta al utilizar el Servicio de Computación para el estudiante, un sistema de computación típico puede aceptar entradas y salidas en una amplia variedad de formatos. Teniendo en cuenta nuestro modelo, supondremos que el supervisor interpreta los datos de entrada y compone los datos de salida.

8.3.9 Laboratorio de programación: Parte 4, entrada y salida de valores numéricos

8.3.9

Tema principal
* * *

En esta sección nos concentraremos en extenso en solo un aspecto del problema de la comunicación con un sistema de computación, a saber, el problema concerniente a la entrada y salida de valores numéricos bajo el control de un programa. Otros aspectos del problema de la comunicación con un computador se discutirán detalladamente más adelante. Utilizaremos el modelo desarrollado en la sección anterior como base de nuestra discusión.

Cuando usted ha programado cuidadosamente una completa sucesión de etapas para ejecutar un cálculo específico, estará naturalmente interesado en conocer el resultado. El resultado estará ahora contenido en uno o más sitios del elemento de almacenamiento, así que, necesitamos de una instrucción para copiar el valor contenido en un sitio de almacenamiento en una tira de papel. La instrucción apropiada se escribe en BASIC en la forma

```
PRINT X
```

Esta declaración se interpreta como sigue:

Etapa	Supervisor	Mensajero
1	Identifica una instrucción de salida; comienza a ensamblar información de salida.	
	1 Envía al mensajero a copiar el valor de X.	
		Va al elemento de almacenamiento, escoge el sitio X, copia el valor en una tira de papel y regresa.
	2 Copia el valor de X en una tira de salida.	
2	Verifica que la lista de salida está completa.	
	1 Envía al mensajero a colocar la tira de salida en el elemento de salida.	
		Va al elemento de salida, deposita la tira y regresa.

Varias variables se pueden sacar por medio de una sola instrucción de impresión; por ejemplo, la instrucción

```
PRINT X, Y, Z
```

dará como resultado una tira de papel de salida que contiene los valores de x, y, y z impresos a intervalos de 15 espacios (con hasta cinco zonas de impresión por línea). Cada instrucción de impresión normalmente hace que la impresión comience en una nueva línea, pero si la

instrucción termina en una coma, entonces la siguiente impresión continuará en la línea antigua. La instrucción

```
PRINT
```

por sí misma se interpreta como "línea alimentada". Si una línea se completa, entonces las instrucciones

```
PRINT
PRINT X
```

darán como resultado dos tiras de salida: la primera en blanco (es decir, con una línea en blanco), y la segunda con el valor de x.

Uno no está restringido simplemente a imprimir el valor de las variables: cualquier expresión, tal como pudiera aparecer al lado derecho de una instrucción de asignación, puede aparecer en una instrucción de impresión; el valor numérico de la expresión total será entonces calculada e impresa. Como un ejemplo sencillo, la instrucción única

```
PRINT 2↑(1/3)
```

da como resultado la salida de la raíz cúbica de 2 (con seis cifras significativas).

Hasta ahora hemos discutido sobre la impresión de los valores de las variables; también es posible imprimir mensajes o encabezamientos escribiendo una instrucción de impresión que contenga como una de sus partes constitutivas en su lista de impresión el encabezamiento apropiado encerrado entre comillas. Por ejemplo, si x tiene el valor 3,5, entonces la instrucción

```
PRINT "X = ", X
```

dará como resultado una tira de salida que contiene la expresión

```
X = 3,5
```

mientras que las instrucciones

```
PRINT "X",  "RAIZ CUADRADA DE X"
PRINT X, X↑(1/2)
```

dará como resultado dos tiras de salida que contienen

```
X          RAIZ CUADRADA DE  X
3,5        1,87083
```

Todos los valores numéricos son correctos con seis cifras significativas. Todos los números muy grandes o muy pequeños son impresos en el formato normalizado de punto flotante con una mantisa de seis dígitos.

Usted puede introducir un valor numérico a una variable x escribiendo una instrucción

```
INPUT X
```

como parte de su programa. Esta instrucción se interpreta como sigue:

Etapa	Supervisor	Mensajero
1	Reconoce una instrucción de entrada	
	1 Envía al mensajero a buscar la siguiente tira de entrada.	
		Va a la unidad de entrada, saca la tira siguiente y regresa.
	2 Copia el valor de entrada en la tira de valores y envía al mensajero a colocarla en el sitio X.	
		Va al elemento de almacenamiento; busca el sitio X; almacena la tira con el valor y regresa.

Tema principal
* * *

Observe que, después de dar tal instrucción de INPUT, el usuario en un terminal deberá recibir de inmediato un signo de interrogación (?) y entonces debe dar cualquier constante numérica escrita en el formato BASIC como ítem de información de entrada. El efecto de la instrucción es dar un nuevo valor a x, siendo este valor dado por el usuario en lugar de escribirlo dentro de una declaración del programa.

El valor de varias variables se puede introducir por medio de una sola instrucción. Así

```
INPUT X, Y, Z
```

producirá un signo de interrogación en el terminal como una petición y entonces debe darle un dato de entrada; usted debe escribir tres números, separados por comas.

Cuando hay más de una instrucción de INPUT en el mismo programa puede suceder que usted no esté muy seguro acerca de lo que debe hacer en respuesta a una petición de datos de entrada. La solución consiste en incluir la instrucción de impresión PRINT inmediatamente antes de cada instrucción INPUT, la cual imprimirá algún mensaje conveniente cada vez. La sucesión de instrucciones:

```
PRINT "ESCRIBA UN VALOR PARA X",
INPUT X
```

dará como salida

```
ESCRIBA UN VALOR PARA X?
```

y espera que se le suministre un dato de entrada.

Resumen

El usuario puede inspeccionar los resultados de un cómputo utilizando la instrucción PRINT. Los valores se pueden introducir en los sitios de almacenamiento por medio de una instrucción INPUT. La ejecución de

dichas instrucciones se puede interpretar en función de un modelo sencillo como elementos de entrada y salida de un sistema de computación.

Ejercicio práctico 4

Ejercicio práctico 4

Usted debe trabajar en este ejercicio cuidadosamente usando lápiz y papel y, a continuación, procese su respuesta en un terminal de computación por medio del programa almacenado DEMON, poniendo

```
GET – $DEMON
RUN
```

En el presente caso el número del ejercicio es 4. Si usted está utilizando el servicio de correo debe presentar sus respuestas claramente escritas en una hoja de codificación en la forma especificada en la *Guía del usuario*.

1 Escriba instrucciones en BASIC para sacar impresos los valores de las variables *a*, *b*, *c*, y *d*;

(i) En una sola línea.
(ii) En columna, una por línea.
(iii) Los valores de *a* y *c* en la línea 1; las líneas 2 y 3 en blanco: los valores de *b* y *d* en la línea 4.

2 Escriba instrucciones en BASIC para leer los valores numéricos de las variables *a*, *b*, *c*, y *d*:

(i) Una a la vez, en respuesta a peticiones individuales.
(ii) Todas a la vez, en respuesta a una sola petición.

3 Describa la ejecución de la siguiente sucesión de instrucciones:

```
INPUT A, B, C
LET S = (A + B + C)/2
LET D = (S * (S – A) * (S – B) * (S – C)) ↑ (1/2)
PRINT "LADOS DEL TRIANGULO", A, B, C,
PRINT "AREA DEL TRIANGULO", D
```

Escogiendo como conjunto de valores de las variables *a*, *b*, *c* los números 3, 4, 5 respectivamente, explique la comunicación entre el usuario y el sistema que debe resultar de la ejecución de esta sucesión de instrucciones (es decir, haga una lista de tres líneas de entrada-salida como si fueran impresas en la hoja de resultados).

4 Especifique la entrada y salida resultantes de la ejecución de las siguientes instrucciones donde B y H tienen los valores 5 y 4 respectivamente.

```
PRINT "BASE, ALTURA"
INPUT B, H
LET V = (B ↑ 2) * H/3
PRINT
PRINT "VOLUMEN ES", V
```

■

8.3.10 Concepto de programa almacenado

8.3.10

Tema principal

En el modelo de sistema de computación que hemos desarrollado anteriormente, la actividad desarrollada por el supervisor está especificada por la sucesión de instrucciones que él interpreta. El supervisor interpreta cada instrucción por turno, en el orden en el cual el usuario las presenta. Hasta el momento no hemos hablado acerca de la manera en que el usuario presenta un programa completo al supervisor. Pensando un poco sobre esto se nos hará evidente que hay tres formas posibles en las cuales esto puede llevarse a cabo, a saber:

(i) El usuario puede darle el programa al supervisor instrucción por instrucción, a través de un elemento de entrada (o bien el mismo utilizado para suministrar datos de entrada, o bien uno separado).
(ii) El usuario puede suministrar el programa en su *totalidad* al supervisor en forma de una sucesión de instrucciones que éste utiliza para determinar sus acciones.
(iii) El usuario da al programa un tratamiento igual al que se da a cualquier otra parte de la información asociada con el problema y actúa en forma tal, que el programa queda almacenado en un conjunto de sitios de almacenamiento.

Todas estas técnicas han sido, y aún lo son, utilizadas como medios de alimentación de un programa a un sistema de computación. Los métodos están colocados en orden de dificultad creciente con respecto a la naturaleza del mecanismo necesario para implementar la técnica. Las dos primeras técnicas son útiles pero ambas presentan limitaciones, así que, concentraremos nuestra atención en la tercera.

Incorporaremos la tercera técnica en nuestro modelo introduciendo los siguientes conceptos adicionales:

1 *Carga y comienzo del programa*
El usuario tiene un "timbre" por medio del cual puede comunicarse con el supervisor. Cuando oprime el timbre, el supervisor ejecuta las siguientes instrucciones:

(i) Detiene la ejecución del programa que está elaborando (si la ejecución no se ha completado).
(ii) Borra toda información (direcciones, valores e instrucciones) del elemento de almacenamiento dándole instrucciones al mensajero de destruir el contenido de cada sitio y quitar las marcas que hay en los sitios de almacenamiento.
(iii) Admite instrucciones, una a la vez, del medio de entrada y las coloca sucesivamente en un conjunto de sitios de almacenamiento tomando nota del sitio en el cual se almacena la primera instrucción.
(iv) Cuando el usuario oprime dos veces el timbre para indicar que ha llegado al final del programa, deja de aceptar instrucciones y comienza la ejecución del programa recientemente cargado, comenzando por la primera instrucción.

2 *Interpretación de un programa*

Cuando un programa se ha almacenado en un conjunto de sitios de almacenamiento, el supervisor interpreta el programa por medio de un ciclo traiga-ejecute que puede ser descrito como sigue:

(i) Cada sitio tiene una dirección (marca) y, por sencillez, supondremos que el conjunto de direcciones es el conjunto de los nú-

meros naturales 1, 2, 3, . . . La dirección de un sitio es, por supuesto, diferente de la variable almacenada en ese sitio.

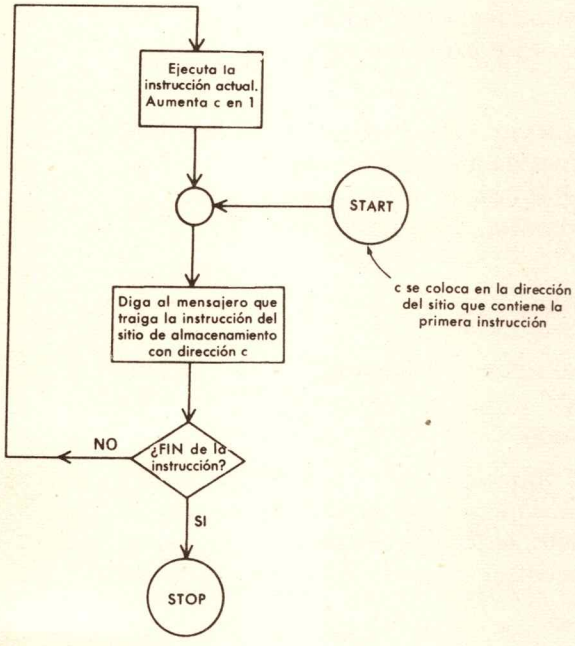
(ii) El supervisor conserva un recuento de la *dirección* de la instrucción que se está ejecutando (o de la primera instrucción mientras carga un programa).

(iii) Cuando la ejecución de la instrucción actual se ha completado, el supervisor envía al mensajero a traer la siguiente instrucción (o sea, la instrucción que está en el sitio con dirección $c + 1$ donde c es el contenido del sitio C cuyo valor es la dirección de la instrucción que acaba de ejecutarse).

Nota:

Suponemos que cada instrucción queda totalmente almacenada en un solo sitio de almacenamiento.

En consecuencia, las actividades del supervisor se pueden resumir en el siguiente diagrama de flujos:



Con esas adiciones, hemos especificado un modelo conceptual para un sistema de computación con el suficiente detalle como para ser capaces de construir programas. Podríamos extendernos agregando nuevos conceptos para aumentar el poder y versatilidad de nuestro sistema modelo con el fin de acercarlo a las características mostradas por un sistema de computación moderno, y también extender la gama de algoritmos para los cuales podemos convenientemente escribir programas. Sin embargo, pospondremos la discusión sobre dichas extensiones hasta que usted haya tenido la oportunidad de usar un sistema de computación y el lenguaje de programación disponible en el Servicio de Computación para el estudiante. Trabajando en problemas prácticos sencillos le dará a usted una oportunidad de estudiar una realización completa de los principios abstractos que hemos discutido y lo ayudará a sentar los fundamentos sobre los cuales podemos construir nuestra futura presentación de conceptos adicionales. Otras consideraciones sobre el concepto de programa almacenado se incorporarán en el programa de televisión asociado a esta unidad.

8.3.11 Laboratorio de programación: Parte 5, ensamblaje de un programa en lenguaje BASIC

8.3.11

Tema principal
* * *

Basándonos en los conceptos adicionales descritos anteriormente, estamos en posición de discutir la construcción de un programa BASIC completo.

Usted ya conoce los ingredientes que se necesitan para hacer un programa sencillo:

(i) Asignar valores a las variables.
(ii) Introducir valores a las variables cuando esto se pide.
(iii) Cálculos utilizando expresiones aritméticas.
(iv) Salida de resultados.

Necesitamos ahora examinar cómo se mezclan estos ingredientes para hacer un programa real.

Un programa consiste de una sucesión de instrucciones; como cualquier otra sucesión, en consecuencia, esta debe comenzar en alguna parte*. Aunque todo programa en BASIC siempre comienza por el principio, en el sentido de que su principio es único, este principio no es necesariamente la primera instrucción.

Definición 1
* *

Introducimos el concepto de *orden* en lenguaje BASIC. Una orden, es una instrucción dada al supervisor la cual se ejecuta de inmediato, y no constituye una parte del programa almacenado. Las órdenes corresponden al timbre de nuestro modelo conceptual. Tal orden, presentada en la forma

SCRATCH

borra el programa actualmente almacenado en el sistema de computación. Esta orden se utiliza para desembarazarse del programa que se ha ejecutado antes de que se inicie un nuevo programa.

En la *Unidad 7, Sucesiones y límites I*, encontró sucesiones que son infinitas, pero usted no las encontrará en programación; por lo menos en lo que concierne a un programa escrito. En BASIC, el final de un programa se debe indicar explícitamente con una instrucción escrita así:

END

Cuando ha sido obedecida, esta instrucción detendrá el programa e imprimirá la palabra DONE; esta instrucción debe presentarse solo una vez en cada programa BASIC.

Para completar, mencionamos aquí que un programa se puede interrumpir en cualquier momento dando una orden de interrupción oprimiendo el botón marcado

BREAK

Esta acción deja completamente de lado al sistema BASIC y es una petición directa al computador para que detenga la ejecución del programa. En general, se le aconseja no utilizar este dispositivo sino como

* "Por favor su Majestad, ¿por dónde debo comenzar?" preguntó él. "Comienza por el principio" respondió el rey gravemente, "y continúa hasta que llegues al final: entonces detente".
Lewis Carroll
Alicia en el país de las maravillas

último recurso: después de que se ha tomado tan severa acción usted encontrará que es necesario recomenzar el programa desde el principio. Las circunstancias bajo las cuales podría considerar correcto el finalizar un programa en forma tan abrupta se expondrán más tarde.

Hasta el momento hemos evitado deliberadamente mencionar una característica particular del BASIC ya que no deseábamos distraerlo de los conceptos fundamentales que expusimos. Sin embargo, al ensamblar un programa completo, esta característica es de importancia fundamental: el principio involucrado es el correcto ordenamiento de las instrucciones que forman un programa. Lo que debemos tener en cuenta es la especificación de la exacta sucesión en la cual las instrucciones deben ser obedecidas: como ya lo sugerimos, esta sucesión no es necesariamente la misma en la cual se escriben las instrucciones en BASIC.

Para aclarar: todas y cada una de las instrucciones en un programa BASIC deben comenzar con un único número. Este número puede ser cualquier entero sin signo comprendido entre 1 y 9999, y es llamado el **número de línea** de la instrucción que encabeza. Ningún par de instrucciones del mismo programa puede llevar el mismo número de línea: si sucede que usted inadvertidamente escribe el mismo número de línea al comienzo de dos o más líneas, entonces todas salvo la última de esas líneas serán ignoradas. Las líneas con órdenes no tienen número de línea, pues las órdenes no se almacenan como parte del programa.

Definición 2
* *

Una idea aparente podría parecer el usar simplemente 1 como número de línea para la primera instrucción escrita, a continuación 2 como número de línea para la segunda instrucción escrita, y así sucesivamente: si usted hace esto, las instrucciones serán evidentemente obedecidas en el orden en que están escritas —entonces, ¿por qué molestarse? Lo que pasa es que la sucesión en la cual las instrucciones en BASIC se ejecutan está determinada no por el orden en que éstas están escritas y presentadas al computador, sino por el orden numéricamente creciente de sus números de línea. En consecuencia, cualquier reordenación de las instrucciones en un programa BASIC no afectará en absoluto el orden en el que esas instrucciones se ejecutan, teniendo en cuenta que cada instrucción conserve su número de línea original. No se sugiere que usted adquiera el hábito de barajar las instrucciones de su programa en forma indiscriminada: entonces, ¿para qué sirve realmente esa comodidad?

La idea general es la de permitirle incluir dentro de su programa, como último recurso, instrucciones que pudo olvidar en una primera etapa. Podría preguntarse cómo es posible introducir otra instrucción entre dos instrucciones existentes con números de línea (digamos) 17 y 18: la respuesta es que, claro está, usted no puede hacerlo. En consecuencia, no es prudente de su parte dar a sus instrucciones números de línea consecutivos en primer lugar: el sistema BASIC no se descarta por números de línea "faltantes". Así que, es buena idea, como primer intento, dar números de línea en múltiplos de 10. Si las instrucciones sucesivas de su programa están numeradas 10, 20, 30, . . ., entonces habrá espacio para que posteriormente usted pueda agregar instrucciones que haya olvidado.

Una vez escrito un programa, usted deseará ensayarlo en el computador. El primer inconveniente que debe vencer es escribir correctamente las instrucciones en la consola*; usted debe acordarse de oprimir la te-

* El número que aparece como 0 en algunas consolas es el número cero y sirve para distinguirlo de la letra O.

cla marcada RETURN después de escribir cada instrucción. Los espacios se ignoran en el lenguaje BASIC, así que es buena idea introducir unos pocos espacios convenientes para mejorar la legibilidad de su programa. No se preocupe si comete un error al escribir una línea: sencillamente vuelva a escribirla correctamente en la línea siguiente; esta línea, que tiene el mismo número de línea, dará como resultado que se ignore la línea equivocada.

¿Cómo es finalmente la situación cuando ha escrito correctamente la totalidad de su programa en la consola? En este punto, su programa estará almacenado en la unidad de almacenamiento del computador. Si está listo a ensayar su programa, puede hacer que el computador comience obedeciendo las instrucciones que usted le ha suministrado, dando la orden

RUN

Observe que esta orden, como todas las demás órdenes, no es parte del programa BASIC y, en consecuencia, no tiene número de línea. El computador comenzará entonces a obedecer su programa e imprimir los valores requeridos por las instrucciones de impresión en el programa. Cuando el computador llega al final de su programa, imprimirá el mensaje

DONE

y esperará sus otras instrucciones.

Bien pronto, usted aprenderá una de las duras realidades de la vida en cuanto a la programación se refiere: sólo muy rara vez los programas se comportan como se supone cuando se ensayan por primera vez. Usted puede ser muy afortunado con los programas sencillos a los cuales se ha limitado en esta ocasión: de hecho ésta es una de las razones principales por las cuales se ejerce un control sobre su progreso. De todos modos, más pronto o más tarde, usted tendrá que encarar el hecho de que uno de sus programas cuidadosamente elaborado simplemente no trabaje y allí es cuando las cosas comienzan a volverse interesantes.

Podremos dar por descontado ciertos errores evidentes sobre los cuales se le ha llamado la atención antes de que usted llegara a la posición de ser capaz de procesar el programa; por ejemplo, si usted no escribe un número de línea de cualquier línea de su programa, el computador rehusará aceptar esa línea y responderá con un mensaje de error.

Otros **mensajes de errores** más explícitos pueden ser impresos por el sistema BASIC, algunos antes de que el programa sea procesado, tales como

Definición 3
* *

ILLEGAL FORMAT

el cual indica que algo hay equivocado en una expresión aritmética —o con la forma con la cual está escrita una instrucción. Tales mensajes están indicados por la salida de la palabra ERROR. Con el fin de conseguir el mensaje de error específico, que indique qué es lo que está equivocado, usted debe oprimir cualquier tecla de la consola seguida por RETURN.

Otros errores se harán evidentes sólo cuando se esté procesando el programa; no todos los errores son detectables por el sistema, pero aquellos que lo son están indicados por mensajes de error tales como

DIVIDE BY ZERO — WARNING ONLY

—aquí, en lugar de dividir realmente por cero, el computador supone que la respuesta es el máximo número que puede contener (aproximadamente 1,70141 E 38) y continúa con el programa.

Otro mensaje de error que puede producirse durante la ejecución del programa es

OVERFLOW — WARNING ONLY

—que se presenta cuando quiera que un cálculo dé como resultado un número con valor absoluto mayor que 1,70141 E 38 aproximadamente.

Le hemos dado sólo unos pocos ejemplos de la clase de mensajes de error que puede encontrar en esta etapa. La lista completa de todos los mensajes de error posibles en el sistema BASIC, junto con sus significados, se da en la *Guía del usuario.*

Algunas fallas no las registra el sistema y, es inútil decirlo, estas son las más difíciles de encontrar. Por ejemplo, aparece un resultado incorrecto: debe resignarse a examinar su programa desde el principio y detalladamente con la esperanza de encontrar dónde estaba equivocado. En el argot profesional, un programa errado se dice que tiene uno o más "bugs"; por esta razón, la tarea de corregir un programa se denota usualmente por el término "de-bugging". Este es un arte que usted sólo puede aprender con trabajo arduo, pero haremos lo que más podamos para ayudarlo. El primer consejo que estamos en capacidad de darle es que usted debe escribir cuidadosamente sus programas cuando los desarrolla, así más tarde usted sabe exactamente qué intentaba que éstos hicieran. En cualquier punto de un programa BASIC usted puede hacer que un número de línea encabece una nota aclaratoria, escrita REM, seguida por cualquier observación que usted desee hacer.

Puede ser que usted no aprecie en su verdadera magnitud la necesidad de tales observaciones mientras esté escribiendo el programa, que aún tiene fresco en la memoria, pero en alguna fecha posterior encontrará que necesitará de toda la ayuda posible para desenredar su propio programa.

Suponiendo que ha logrado encontrar un error en su programa, ¿cómo haría para corregirlo? Si se trata de remplazar una instrucción errada, sencillamente escriba el número de línea de dicha instrucción seguida por la versión corregida. Esto hará que el computador ignore la versión incorrecta anterior de esta instrucción particular. De la misma manera puede cambiar cualquier número de instrucciones erradas.

Usted podría desear remover del programa una instrucción particular: esto puede hacerse escribiendo su número de línea seguido por una línea en blanco. Las líneas en blanco, así como los espacios, nunca se tienen en cuenta.

Si inicialmente asigna los números de línea en múltiplos de 10, no tendrá dificultad en incluir instrucciones extras en las instrucciones existentes de su programa; todo lo que necesita hacer es escribir las instrucciones extras, precedidas todas por el número de línea apropiado para indicar su lugar en el programa.

Cuando ha corregido el programa de esta manera, podría desear ver cómo luce ahora. Si escribe la orden

LIST

la versión corregida de su programa será impresa en la consola. Si está satisfecho con ella, puede ensayarla de nuevo dando la orden

RUN

Es de esperarse que debe ejecutar este procedimiento varias veces antes de que su programa trabaje satisfactoriamente.

Concluyamos con un ejemplo sencillo.

Ejemplo 1

Considere el problema de determinar la verdadera rata de interés anual sobre un préstamo de compras.

Sean c el precio de costo de los bienes,
d la cuota inicial,
t el número total de pagos para cancelar el préstamo,
p el número de pagos en un año,
y h interés sobre la compra total.

Puede demostrarse que la verdadera rata de interés r está dada por la fórmula

$$r = \frac{200ph}{(c - d)(t + 1)}$$

La siguiente sucesión de instrucciones (impresas en negro) dadas a una consola llevará a cabo este cómputo. El material impreso en rojo será dado como impresión de salida por el sistema de computación. Por ejemplo, después de que la instrucción con número de línea 20 se ha ejecutado, el usuario será requerido por una? y entonces debe suministrar los números 42, 8, 8. (Las palabras impresas en itálicas son simplemente explicativas y no forman parte ni de las instrucciones ni de la impresión de salida.)

```
SCRATCH   inicia al sistema (orden)
 10  REM... CALCULE VERDADERA RATA DE INTERES...
 20  PRINT "PRECIO COSTO, DEPOSITO, TOTAL INTERESES
     COMPRA TOTAL",
 30  INPUT C, D, H
 40  PRINT "PAGOS ANUALES, NUMERO TOTAL PAGOS",
 50  INPUT P, T
 60  REM... DATOS DE ENTRADA HAN SIDO ENSAMBLADOS
 70  LET R = 200 * P * H/((C - D) * (T + 1))
 80  REM... SE HA COMPUTADO LA VERDADERA RATA DE
     INTERES
 90  PRINT "VERDADERA RATA DE INTERES ES", R
100  REM... EL CALCULO ESTA COMPLETO
110  END
```

RUN *ejecute el programa (orden)*

PRECIO COSTO, DEPOSITO, TOTAL INTERES COMPRA TOTAL? 42, 8, 8

PAGOS ANUALES, NUMERO TOTAL PAGOS? 12, 12

VERDADERA RATA DE INTERES ES 43,4389

DONE

programa completo (mensaje del sistema)

Encontrará útil estudiar este ejemplo cuidadosamente antes de intentar resolver los ejercicios siguientes.

Resumen

En el lenguaje de programación BASIC, existen órdenes que permiten al usuario ensamblar y ejecutar un programa completo. La ejecución de dichas órdenes puede interpretarse en función del concepto de programa almacenado. Nuestro estudio del sistema BASIC, sin embargo, no es completo, ya que no sabemos aún cómo interpretar la información estructural de un diagrama de flujos en función de las instrucciones en BASIC. Estudiaremos este problema en el siguiente conjunto de textos de laboratorio.

Ejercicio práctico 5

Ejercicio práctico 5

Debe trabajar en este ejercicio cuidadosamente con lápiz y papel y, a continuación, debe procesar sus programas en un terminal de computación o presentarlos claramente escritos en una hoja de codificación si está utilizando al servicio de correos.

1 Escriba un programa completo que acepte como datos de entrada un par de números y que produzca como salida la suma, la diferencia, el producto y el cociente de dichos números.

2 Escriba un programa que acepte como dato de entrada un entero decimal que represente una suma de dinero en libras esterlinas en la forma £.p e imprima a la salida la cantidad original y la misma cantidad convertida en dólares en la forma $.¢. (Suponga que la rata de conversión es variable.)

(£.p significa libras esterlinas y peniques.)

3 Escriba un programa que dé como resultado el siguiente diálogo entre el usuario y el sistema

PRECIO DE COSTO? *c*

GANANCIA ANUAL? *i*

RENDIMIENTO DE ACCIONES *y*

donde y se calcula a partir de la fórmula $y = \frac{100i}{c}$. El material impreso en rojo es información de salida; c, i, y y son valores numéricos.

4 Corrija los errores que se presenten en el siguiente programa donde los valores de las variables c, k se calculan a partir de la variable f por medio de las fórmulas $c = \frac{5}{9}(f - 32)$, $k = c + 273$. A continuación diseñe el programa para cualquier valor de F con el fin de ob-

tener los correspondientes valores de c y k, y verifique sus resultados utilizando las fórmulas dadas.

```
PRINT "VALOR FAHRENHEIT"
INPUT F
LET C = 5/9 * F - 32
LET K = C + 273
PRINT F,
PRINT C K
```

■

0-07-090767-6